TCP Stretch Acknowledgements and Timestamps: Findings and Implications for Passive RTT Measurement [*]

Hao Ding
University of Science and Technology Beijing
haoding8724@gmail.com

Michael Rabinovich
Case Western Reserve University
michael.rabinovich@case.edu

ABSTRACT

This paper examines several TCP characteristics and their effect on existing passive RTT measurement techniques. In particular, using packet traces from three geographically distributed vantage points, we find relatively low use of TCP timestamps and significant presence of stretch acknowledgements. While the former simply affects the applicability of some measurement techniques, the latter may in principle affect the accuracy of RTT estimation. Using these insights, we quantify implications of common methodologies for passive RTT measurement. In particular, we show that, unlike delayed TCP acknowledgement, stretch acknowledgments do not distort RTT estimations.

1. INTRODUCTION

Round-trip time delay (RTT) is a fundamental characteristic of the Internet communication affecting numerous aspects of network operation, from provisioning and traffic engineering to construction of wide-area network applications. Passive RTT measurements, where RTT values are inferred from observing the actual network traffic, promise non-intrusive and potentially the most faithful approach to RTT measurement. This paper considers several common techniques of passive RTT measurement from methodological perspective and examines several TCP characteristics and their effect on these techniques. In particular, this study makes the following contributions (the extended version of this paper [7] includes additional results).

- We find significant use of so called *stretch TCP acknowledgements* [17], which acknowledge more than every other segment. Although stretch ACKs (or ACK filtering/thinning, which produces them) have been proposed for various environments (e.g., [6, 8, 16]), to our knowledge, ours is the first study to measure their actual usage on the Internet.
- We measure the prevalence of TCP timestamp option and the patterns of TCP acknowledgements. Despite previous studies' findings that over 75% of servers support TCP timestamps [20], or over 92% of flows originating from customers of a particular LTE network operator carry them [11], we find their use in our actual traffic to be low (28–41% of flows). Limited timestamp usage affects the applicability of some measurement techniques as discussed later.
- We quantify the effect of delayed and stretch acknowledgements on accuracy of RTT estimation and find that delayed ACKs do overestimate RTTs while stretch ACKs do not. Prior work noted – but did not evaluate – the effect of delayed acknowledgements [3] on RTT estimation. To our knowledge, we are the first to consider the effect of stretch acknowledgements.
- We confirm prior studies [3] that found that RTTs measured in the middle of a connection tend to be higher than the RTT during the handshake. We further identify hosts connecting through middleboxes (wifi gateways and VPN servers) as the likely reason for this RTT inflation.
- Finally, using passive RTT measurement techniques unaffected by the above biases, we show that local delays within campus intranet, although sometimes neglected in RTT measurements (e.g., [3, 18]), may constitute a substantial portion of the total network delay. We again pin much of the blame on the middleboxes for this behavior.

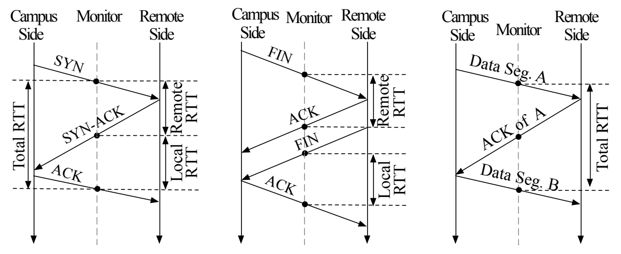
[*]Work supported in part by NSF grant CNS-0831821. The work of Hao Ding was done during his visit at CWRU, which was sponsored by China Scholarship Council.

While we consider implications of stretch acknowledgments on RTT measurement, their significant presence can have far-reaching implications for TCP performance in general, since TCP senders often regulate sending rate based on the number of received acknowledgements [5]. Our study takes but the first glimpse into this phenomenon from two vantage points. We hope this will prompt further research to investigate this phenomenon in greater detail.

2. PASSIVE RTT MEASUREMENT

Passive RTT measurement techniques fall into two broad categories - those that require bidirectional packet trace and those that only need to see packets in one direction. Our setup allows us to capture bidirectional traffic and we mostly focus on the bidirectional techniques. Bidirectional techniques often allow estimation of both total RTT between the communicating end-hosts as well as RTT components from either host to the measurement monitor. We refer to the round-trip time between the campus side and the measurement monitor as local RTT, and between the monitor and the external side as remote RTT. Techniques also differ in whether they allow a single readout per TCP connection (we call them *one-time techniques*) or repeated readouts in the course of the connection (*continuous techniques*). We consider the following techniques in this study.

The popular *handshake* technique utilizes segment association during the TCP three-way handshake. Figure 1(a) illustrates this technique for the case when the caller is from the campus side (but the technique equally applies to the connections initiated from outside). With unidirectional traces, handshake method can only measure the total RTT and only for connections in which the caller-to-callee direction is the one captured. With bidirectional flows, this method can measure all three kinds of RTTs – local, remote, and total – for all connections.



(a) Handshake (b) Teardown (c) Timestamp

Figure 1: Passive RTT estimation methods

A similar method utilizes segment association during the closing of a TCP connection, to which we refer as *teardown* technique. Figure 1(b) shows how this technique measures local and remote RTT on the example of the campus host performing active close. The combined RTT is calculated as the sum of the local and remote RTT. Frequently abnormal TCP closing and various flavors of simultaneous close make this method less applicable than the handshake method.

The above two methods are *one-time* techniques—they can only measure the RTT at the beginning and ending of a connection. Several *continuous* techniques are based on the segment sequence shown in Figure 1(c), where the transmission of segment B is triggered by the arrival of ACK for segment A. While associating data segment A with the corresponding ACK can be done by sequence numbers, the key is in reliably associating data segment B with the ACK that triggered it. Previous studies used TCP congestion window modeling for this purpose, either during slow-start [13] or congestion avoidance [12] phase. The TCP *timestamp* method [20] utilizes TCP timestamp option to associate a data segment with the ACK that triggered it. This method associates segment B and the ACK of A by matching the timestamp in the ACK packet with the echo timestamp echo in data segment B. These methods can measure RTT throughout the lifetime of a TCP session as long as the TCP connection enables timestamp option. Because congestion window modeling is brittle due to differences among various TCP stack implementations, we concentrate on the timestamp method for continuous RTT measurement.

Another popular continuous method, which can only produce either local or remote RTT for a given segment sequence, measures the time difference between a data segment and its corresponding ACK. Since typically the data mostly flows in one direction, this method—to which we refer as *Data-ACK*—produces mostly either local or remote, but not the combined, RTT for a given connection.

An interesting unidirectional technique measures combined RTT as the time between successive data transmission bursts, leveraging the TCP tooth-like sending pattern [20]. It utilizes discrete autocorrelation function to detect the gaps between TCP bursts. However, the algorithm to detect the burst-gap patterns requires several predefined thresholds that are difficult to determine, especially when the network conditions are not sufficiently stable. We do not consider it in this paper.

3. METHODOLOGY AND THE DATASET

This study uses three datasets. Two are TCP packet traces captured at the edge of two campus networks, Case Western Reserve University (CWRU) in Cleveland and Colorado State University

Table 1: Statistics of three traces

Trace	Time of Capture	Packets	Connections		Packet Loss
			All	Valid	
CWRU	Jan 5, 2015 3–7pm	1.5B	17.4M	9.6M	0.4%
CSU	Jan 16, 2015 12–1pm	0.4B	2.8M	2.5M	0.1%
CAIDA	Mar 20, 2014 7–8am	0.9B	14.5M	12.0M	0.1%

(CSU) in Fort Collins, CO [2], and the third is a TCP packet trace of the traffic on a backbone link between Seattle and Chicago provided by CAIDA [1][1] All captures are done using Endace 10Gbps cards. Because the CSU and CAIDA traces are host-anonymized, we could only use them for some of our analysis verifying our key findings; the rest of the analysis is done on the CWRU trace. Moreover, while the CWRU and CSU traces are bidirectional, the CAIDA trace typically sees packets in only one direction in a given connection due to asymmetric routing, and in fact we only use one direction (Seattle to Chicago) in our study. Thus further limits the questions we could answer using the CAIDA trace.

Table 1 shows the trace statistics. We split the traces into TCP connections by first putting all packets with the same end-point IP addresses and ports into separate bucket and – in case multiple connections reuse the ports – further splitting each bucket by locating TCP handshakes within each bucket (which happens in only less than 2% of the buckets in each trace). We consider a connection valid if it includes at least one segment carrying the ACK flag but not the SYN, FIN or RST flags. Only valid connections are used in our analysis. The CWRU trace is bidirectional, the CAIDA trace is mostly unidirectional, and the CSU trace is bidirectional for most hosts except for a few local prefixes whose traffic is consistently captured in one direction only (the CSU colleagues who provided the trace confirmed that routing setup for these prefixes bypassed the capture point on the inbound traffic). These unidirectional prefixes affect 2.9M packets and 550K valid connections. We use the full trace for the timestamp analysis but only the bidirectional part for the stretch ACK analysis.

To estimate packet loss *in the measurement instrumentation*, we note that, unlike losses in the network, which result in eventual retransmissions and thus manifest themselves as reordered packets in the trace, losses in the measurement instrumentation are not retransmitted and thus never appear in the trace. Consequently, we first identified the gaps in valid connections that were not eventually filled completely by reordered packets. These gaps affected 1.8% connections in CWRU, 0.4% in CSU, and 0.5% in CAIDA traces. We then conservatively consider all the residual gaps to be due to packet loss within the instrumentation and estimate this loss rate by assuming that each residual gap smaller than MSS (which constituted a majority of the gaps) contained one lost packet and covering larger residual gaps with MSS-sized packets. The resulting packet loss is listed in Table 1. Given the small number of affected flows and we still obtain RTT samples from unaffected parts of the flow, the number of flows that could not produce RTT samples is negligible and does not affect our findings. To avoid distorted RTT readout due to retransmitted or reordered packets, we do not use retransmitted data packets or any data packets between retransmit-

[1]The CWRU capture was reviewed by the IRB and found exempted. The other datasets were obtained using the approved procedures from the providing organizations.

Table 2: OS distribution of flow sources (CWRU trace)

	Flows	Windows	Linux	Mac	Unkwn
Local clients	6.8M	70%	10%	18%	1%
Local servers	2.3M	26%	58%	<0.1%	15%
Local total	9.1M	59%	22%	13%	5%
Remote clients	2.3M	54%	34%	8%	3%
Remote servers	6.8M	5%	49%	<0.1%	45%
Remote total	9.1M	17%	45%	2%	35%

ted data packets or between the start of a gap and a packet filling the gap, or duplicate acknowledgements, to generate RTT samples.

Table 2 lists further characteristics of the CWRU trace. Valid connections in the trace represent 10K internal and 359K external IP addresses and 1.2M host-pairs (or more precisely, address-pairs). We used the MaxMind GeoIP database to map the external addresses to 11K autonomous systems in 228 countries/regions. We also used the *p0f* tool[2] to consider the distribution of the operating systems (OSs) on both sides of all valid connections that started with a proper handshake, which accounted for 95.7% of valid connections (the rest included 0.8% that started before our capture, 2.8% that showed retransmission of handshake segments, and 0.6% with missing handshake segments). The table omits OSs p0f could detect but found to be not Windows, Linux or Mac; these represent less than 0.1% of connections on either side.

4. STRETCH ACKNOWLEDGEMENTS

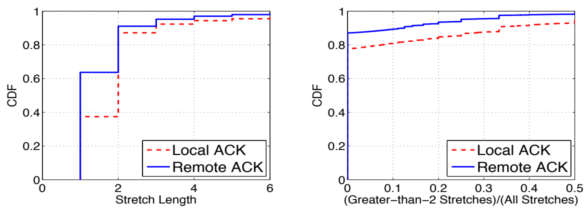
TCP expects a recipient to acknowledge at least every other segment, and this assumption is engrained in both networking practice (e.g., in common congestion control implementations driven by the number of ACKs rather than the amount of data they acknowledge [5]) and TCP-related research (e.g., [12]). However, we find that flows often have fewer acknowledgements, with an ACK cumulatively acknowledging more than two data segments at once (such ACKs were termed "stretch ACKs" in [17]). This finding is interesting because, although there were a number of proposals to use stretch ACKs in various asymmetric environments (e.g., [6,8,16]), RFC 5690 noted in 2010 that the authors were "not aware of the deployment of ACK filtering in the Internet" [9].

To measure the prevalence of stretch acknowledgements, for each connection we measure the length of the data packet trains (we call *stretches*) between consecutive cumulative ACKs (i.e., without an ACK with intervening acknowledgment numbers[3]). For example, if every data segment was acknowledged with a separate ACK, the corresponding stretches would be of length 1. For regular delayed ACKs, with every two data segment being acknowledged cumulatively by one ACK, the corresponding stretches have length 2. Stretch ACKs acknowledge stretches of data segments with length greater than 2. We refer to the data segments as *unacknowledged* if they are only acknowledged cumulatively by an ACK that also acknowledges a later segment. In other words, a data segment with sequence number s and payload size n is called acknowledged if it is followed in the trace with an ACK from the opposite direction with the acknowledgement number $s+n$ and unacknowledged otherwise.

Again, we exclude potential distortions by excluding stretches containing out of order data packets or bracketed by duplicate acknowledgements. Further, an RST segment resets the beginning of a stretch. In what follows, we call a flow local or remote depending on the source of the flow. For instance, a "local flow with stretch ACKs" refers to a flow from a local sender that contains ACK packets acknowledging more than two data packets from the opposite direction.

[2]http://lcamtuf.coredump.cx/p0f3/

[3]For practical reasons, we only look for ACKs within 300 packets from a given data packet.



(a) Stretch lengths (b) Stretch ACK frequency in a flow

Figure 2: Prevalence of stretch ACKs in CWRU trace

4.1 Prevalence

Figure 2(a) shows the distribution of stretch lengths in the CWRU trace. Since at least three data packets are needed for the receiver to exhibit stretch ACKs, when measuring prevalence of stretch ACKs in a given flow, we only consider valid connections with least three data packets coming from the other side. As the result, we inspect 4.1M local flows and 3.5M remote flows in 4.9M connections, including 338.7M samples of stretches (241.9M local and 96.8M remote stretch samples). While most stretches have 1 or 2 segments, 11% of all stretches, including 13% local samples and 9% remote ones, are longer. Such a significant longer stretch prevalence cannot be explained by packet loss in our instrumentation; indeed, this would require at least one ACK packet loss for each stretch over 2 packets, or at least 11% loss among ACK segments. Given our estimated loss rate of both data packets and handshake packets listed in Section 3 (the latter especially indicative as it reflects the loss in both directions), such a high loss rate of acknowledgments is improbable.

We next consider how often individual connections exhibit stretch ACKs. Figure 2(b) plots the distribution of fractions of greater-than-2 stretches (out of all stretches) in each direction. It shows that when a flow uses stretch ACKs, it is not an abberation. In the rest of this paper, we regard a flow to use stretch ACKs if more than 10% of stretches from this flow are greater than 2. It can be seen from Figure 2(b) that 19% of all local flows, and 11% of all remote flows, used stretch ACKs (these percentages correspond to the fractions of flows with ratios over 0.1 on the x-axis in Figure 2(b)).

Our overall conclusion is that stretch ACKs are fairly common. We verify this finding on the CSU trace, inspecting the total of 1.9M flows with at least three data packets in the other direction (we are unable to distinguish local or remote flows due to anonymization). The distribution of the fractions of greater-than-2 stretches is illustrated in Figure 3, which shows that 22% of flows exhibit stretch ACKs – even greater prevalence than in the CWRU trace.

4.2 Possible Causes

Stretch ACKs may be an artifact of the TCP implementation at the end-host or the result of *ack filtering* within the network, which

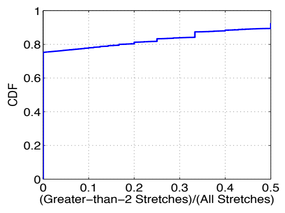


Figure 3: Per-flow stretch ACK frequency in CSU trace

Table 3: OS Distribution of sources of flows with stretch ACKs

	Flows	Windows	Linux	Mac	Unkwn
Local Sources	781K	69%	16%	11%	3%
Remote Sources	371K	17%	39%	3%	40%

has been suggested as the means to reduce router transmission queues and improve TCP throughput in networks with asymmetric links [6,16]. In an attempt to find the source of stretch ACKs, we use the p0f tool to determine the operating systems of all sources of flows with stretch ACKs (only the connections that started with proper handshake were considered). From Table 3 we see that both local and remote sources of stretch-ACK flows come from a variety of operating systems, and the distribution of OS types is roughly similar to the OS type distribution for all the flows, shown in Table 2. This provides some initial indirect evidence against the end-hosts being the culprit (since stretch ACKs are not limited to a particular TCP/IP stack implementation), leaving the network elements as a more likely cause. Section 6 below provides further – and stronger – support to this conjecture.

Table 3 does show some increase in the prevalence of Windows among local sources of stretch ACK flows. The reason for this increase is unclear. There is a parameter "TCPAckFrequency" in Windows that allows setting the frequency of TCP acknowledgements. But since the default value is no greater than 2 in all Windows versions, we believe it is unlikely that many users change it.

Table 4: Timestamp Option Prevalence

	TCP Packets	Valid Connections
CWRU	33%	28%
CSU	27%	23%
CAIDA	57%	41%

5. TCP TIMESTAMPS

Previous studies that relied on TCP timestamps for passive measurements provided indirect indication of their common use: [20] and [14] note that, respectively, over 75% and 83% of the servers they probed supported the timestamp option, [11] report that over 92% of flows originating from customers of a particular LTE network carry them, and [19] states that "the TCP Timestamp option is known to be widely deployed in the wild."

We check the TCP timestamp prevalence in actual traffic, using the three datasets studied. Table 4. shows the fraction of all packets (including from invalid connections) that carry timestamps, and the fraction of all valid connections using them. All traces, especially CWRU and CSU, show lower timestamp use than the levels implied in previous studies.

Table 5: OS of CWRU clients without timestamp support

	Flows	Windows	Linux	Mac	Unkwn
Local Clients	4.8M	98%	0.4%	<0.1%	1%
Remote Clients	1.3M	95%	3%	<0.1%	1%

To see if this is attributable to clients or servers, or any specific TCP stack, we conduct two experiments. First, we check the timestamp option announcement in the SYN segments for all the valid connections that started with a proper handshake of CWRU trace (9.1M connections), and find out that only 29% local client flows and 43% remote client flows announce the timestamp option in the SYN segment. We further use p0f to determine the OS distribution of the clients that do not support timestamp option, shown in Table 5. It can be seen that Windows clients disproportionately contribute to the low prevalence of timestamp use. Specifically, only 9.2K local windows clients (0.2% of all local windows clients) and 1.1K remote windows clients (0.9% of all remote windows clients) supported timestamps[4].

We next consider the prevalence of timestamp support among servers. According to RFC 1323, the server can not announce timestamps in its SYN-ACK segment unless it received the timestamp option in the SYN segment from the client. Thus, we only inspect the servers that receive timestamp option announcement from the clients. We find 97% of local servers and 84% remote servers support timestamps. Therefore, our study actually supports the previous studies [14,20] in terms of the wide support of TCP timestamps among servers but not the implication (such as expressed in [19]) that this indicates their wide use.

To our knowledge, the only study that passively measured TCP timestamp use is [15] conducted 10 years ago, which found 21.5% of clients announcing timestamp support (an even earlier study [4] measured timestamp use from one website perspective, which could be biased by the clients accessing the site). Our data shows their use remains rather low, which limits the applicability of passive measurement techniques relying on TCP timestamp presence.

6. COMPARISON OF RTT MEASUREMENT TECHNIQUES

We compare RTT measurement techniques in this section. We round our RTT samples to 1 millisecond precision, and those under 1ms are recorded as 1. To compare two given RTT measurement techniques, we present the cumulative distribution of the ratios of connection RTTs derived using both techniques. This allows us to detect any biases between the techniques involved but not absolute differences. We include results on absolute differences in the extended version of the paper [7].

6.1 Handshake vs. Teardown RTT Inference

A frequent concern expressed with the handshake technique is that it may overestimate RTT since it includes server processing to initialize the connection. Thus, we compare the handshake and teardown methods for connections where both phases are well-formed. We find that only around 20% to 25% of connections have

[4]These numbers do not match those one could derive from Tables 2 and 5 due to rounding.

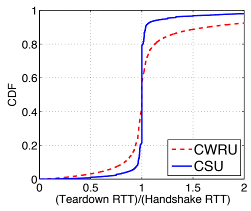


Figure 4: Teardown to handshake RTT ratio

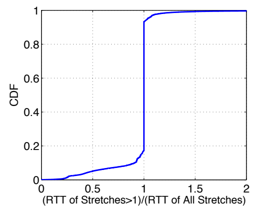


Figure 6: Delayed ACK effect on Data-ACK RTT

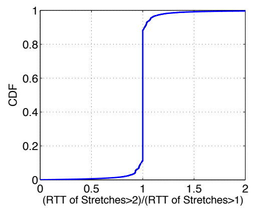
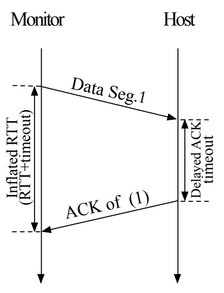
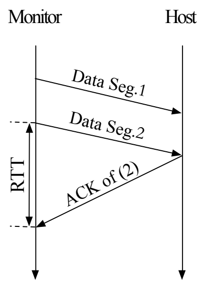


Figure 7: Stretch ACK effect on Data-ACK RTT



(a) RTT inflation due to delayed ACK timeout



(b) Delayed ACK effect eliminated

Figure 5: Eliminating RTT inflation from delayed ACKs

a close that allows unambiguous teardown estimation[5]. Figure 4 plots the CDF of ratio of the teardown to handshake estimates of full (i.e., including both local and remote) RTT for CWRU trace (2.3M data points) and CSU trace (479K data points), for connections that provide both estimates (each connection contributes one data point). While in most cases there is a variation between the two methods, one metric exceeds the other in almost half of the cases with difference less than 1%, which indicates that any processing at the server in handshake estimates does not lead to consistent RTT overestimation relative to the teardown method.

6.2 Effect of Delayed Acknowledgements

The Data-ACK and timestamp methods of RTT estimation can be skewed by delayed acknowledgements[6]. For example, in the Data-ACK method, when a data segment arrives and the receiver has already acknowledged all the previously received data, the acknowledgement can be delayed by up to 500ms thus inflating the RTT estimate (Figure 5(a)). However, in practice, the delay threshold can be an order of magnitude lower, and then there are various scenarios when acknowledgments are not delayed at all. We would like to understand if ignoring delayed acknowledgements biases estimates in practice. Since the timestamp method applies to fewer connections and is biased towards non-windows end points (see Section 5), we consider the Data-ACK method here; further, because the data-ACK method is one-sided (i.e., measures either remote or local RTT), we concentrate on remote RTTs as they are a typical focus of measurements, where the delayed ack effect is sometimes neglected (e.g., [3]).

To assess RTT inflation, we apply two flavors of Data-ACK estimation to the CWRU trace. First, we derive a sample from every pair of a data segment and its ACK based on the matching sequence numbers, which may include inflated RTT samples as illustrated by Figure 5(a). Second, we only collect a sample if the data segment participating in the estimate follows an unacknowledged data segment (thus eliminating a possibility of the delayed acknowledgement bias, since the delayed ACK mechanism stipulates that the arrival of the second data segment triggers an immediate ACK, see Figure 5(b)). Then, for each TCP connection with at least one RTT sample of both kinds, we compute the ratio of the median values of RTT samples that ignore and remove the delayed ACK impact.

Figure 6 shows the CDF of these ratios of 1.5M connections of the CWRU trace. As we see, delayed ACKs do inflate the RTT estimation: indeed, median RTTs with removed delayed ACK impact exceed those that do not in 6% of connections and are smaller in around 17% of connections, although more than 75% of the ratios have value 1. Thus, if one is interested in a single connection, ignoring delayed ACKs will likely still produce the unbiased result. However, cumulative analysis of a large number of connections will result in a bias.

6.3 Effect of Stretch Acknowledgments

Similar to delayed ACKs, stretch ACKs, can inflate RTT measurements if a host waits for a certain number of data segments before sending an acknowledgement. We are interested if stretch ACKs further inflate RTT beyond the "normal" delayed ACKs considered in Section 6.2. To this end, we collect all remote RTT samples that exclude the effect the delayed ACKs (as described in Section 6.2) and all samples where the data segment participating in the estimate follows at least *two* unacknowledged segments. The latter samples eliminate RTT inflation from stretch ACKs if a host acknowledges at least every third data segment but not if the host allows longer stretches. Although our assessment underestimates the total amount of inflation, it captures the largest jump (if any): eliminating longer stretches would drastically reduce the number of available samples but could only capture the diminishing inflation increments.

Figure 7 presents the CDF of the ratios of the median values of the above two kinds of sample RTTs for all 328K connections with stretched ACKs from the remote side (i.e., whose remote flows showed at least 10% of stretch ACKs) that produced at least one sample of each kind. The graph shows no inflation effect, with roughly 80% of the ratios equal to 1 and the rest split evenly (within 0.5%) between over- and under-estimation. We conclude that our data shows no effect of stretch ACKs on RTT estimation. This also provides further evidence that hosts are not the source of stretch

[5]The low percentage of connections allowing teardown RTT estimation is mostly due to flows that involve RST in the closing phase and, to a less extent, with simultaneous close. A small further fraction of connection also included retransmitted packets in the closing phase. RTT can only be derived reliably from a "perfect" and not-simultaneous close.

[6]Note that unlike TCP's estimation of *effective RTT*, which intentionally includes any delayed acknowledgment effects to avoid premature retransmissions, we are interested in *actual* RTTs.

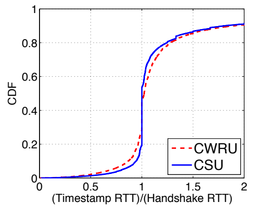


Figure 8: Timestamp to handshake RTT Ratio

ACKs observed in the traces, leaving in-network filtering as the likely reason.

6.4 Timestamp vs. Handshake RTT

We now compare two methods suitable for estimating total RTT in unidirectional flows: the handshake and timestamp methods. We do not consider the teardown method here because it produces similar estimates to handshake (Section 6.1) but offers fewer samples. We use the CWRU and CSU traces; we could not use the CAIDA trace because its unidirectional flows do not allow for the elimination of the effect of delayed ACKs. For the timestamp method we exclude the impact of delayed ACKs by only collecting a sample when the data segment involved follows an unacknowledged data segment. Further, a timestamp RTT sample can be distorted when the sender on Figure 1(c) does not immediately have data segment B to send, which could happen especially often in interactive applications, where the application would send segment B only after the user enters another request. To remove these cases, we exclude timestamp RTT samples over 2000ms, with the intuition that most network delays are less than, and user think time greater than, this value. In the extended version of this paper [7], we show that these filtered timestamp RTT measurements, when applied to only remote RTT, match closely RTTs measured with the Data-ACK method (which is not subject to this distortion), indicating that our filtering successfully mitigates the above distortion.

For each connection that produced at least one estimate of both kinds (349K and 73K in the CWRU and CSU traces respectively), we compute the ratio of the median RTT value according the timestamp method to the (only) estimate according to handshake method. Figure 8 plots the CDF of these ratios[7]. It shows that the timestamp method, which produces measurements throughout the connection, generates larger estimates than those from the handshake method more often (55% of connections in CWRU trace and 46% in the CSU traces) than smaller estimates (only about 27% of connections in CWRU trace and 19% in the CSU trace).

It has been noted previously that RTTs measured in the course of a connection tend to be higher than during handshake [3] but those measurements did not eliminate the delayed ACK impact. We show the bias remains, at least for the timestamp method, even with this impact removed. The rest of this section uses the CWRU trace to shed light on the reasons behind this phenomenon.

Our first question is whether the inflation is due to local (within the campus network we monitor) or remote delays. Since segment exchanges in both timestamp and handshake methods allow one-sided RTT measurements, Figure 9 plots the CDFs of the timestamp-to-handshake RTT ratios of both local and remote sides.

The local timestamp RTTs exhibit higher skew than remote RTTs. Local timestamp RTTs exceed those from handshake in 27%

[7]Note that this distribution has heavy tails and in fact sizable fraction of ratios extend beyond the cut-off value 2 in the graph. Because we are mainly interested in the skew of the estimates rather than maximum errors, this and similar later graphs suffice.

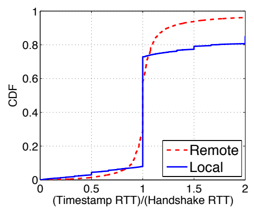


Figure 9: Timestamp to handshake RTT ratios for local and remote RTTs

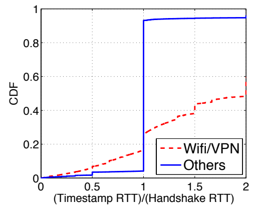


Figure 10: Timestamp to handshake local RTT ratios per connection types

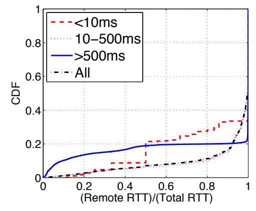


Figure 11: Remote-to-total RTT ratios using handshake estimates

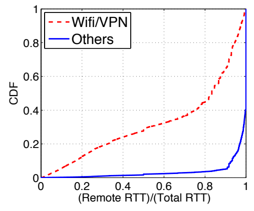


Figure 12: Remote to total RTT ratios for Wifi/VPN hosts and other hosts

cases and only in 8% cases are lower. For remote RTT, in 41% cases timestamp RTTs are higher and in 28% lower than handshake RTTs – showing significantly less inflation than total RTTs. This indicates that local RTTs are indeed largely responsible for RTT inflation.

We speculated the above difference could be due to the fact that most flows (two thirds, see Table 2) in the CWRU trace had local hosts acting as clients, and clients are more likely to connect through middleboxes such as wireless gateways or VPN servers. Thus, knowing the IP addresses of these devices, we considered local RTTs for the flows with local hosts behind these devices separately from the flows with other local hosts. Figure 10 presents the corresponding CDFs of the RTT ratios. Indeed, it shows that virtually all the skew observed in local RTTs in Figure 10 is due to the hosts behind the middleboxes with 74% of their timestamp RTTs inflated. For the other flows, the vast majority (89%) of their local median timestamp RTTs were equal to, and only 7% were higher than their handshake counterparts, while 4% were lower.

One reason why middleboxes might impose longer delays on segments inside the connection than on the handshake segments could lie in their difference in size. Shorter control segments have less chance to encounter interference and experience link-layer retransmission. Another possibility is that arriving window-fulls of segments from inside a connection pile up at the middlebox and queue behind each other, in addition to competing with packets from other connections, while control segments always arrive in isolation and only have to compete with packets from other connections. Further research is needed to fully understand this phenomenon.

6.5 Effect of Local RTT

Finally, we consider the contribution of local RTT to the overall delays, which is sometimes neglected in RTT measurements (e.g., [3,18]). We use the CWRU trace of this analysis as the other traces are anonymized and do not allow determining which side is local. We use the handshake methods for this purpose as it allows derivation of remote and total RTTs depending on the segments considered (see Figure 1). Figure 11 plots CDFs of the ratios of remote to total RTTs for 9.1M connections with handshake samples. According to Figure 11, for about 12% of all samples, the remote RTT accounts for no more than 80% of the total RTT. Further, sizable contribution of local delays is not limited to short-haul communication: the samples in the 10–500ms range follow the same distribution. Short-haul communication does exhibit higher contribution of local RTTs (28% of samples with at least 20% local contribution). For RTTs exceeding 500ms, the remote-to-total RTT ratios are more clustered towards 1 but in 20% of the cases these delays are *primarily* due to local RTT (note that only 0.9% of all RTTs were this high).

To understand the reason for high local RTTs, we split the samples involving local middleboxes from the rest of local hosts. Figure 12 plots the corresponding CDFs of remote to total RTT ratios (1.7M samples for Wifi/VPN hosts and 7.4M samples for other hosts). These plots show that middleboxes are largely responsible for high local RTTs. While the specific mechanisms behind this are outside the scope of this paper, *bufferbloat* [10] has been implicated in excessive network delays.

Our overall conclusion is that local RTTs can add sizable contribution to the overall delay and hence to inaccuracy in measurements ignoring them. However this inaccuracy is mostly limited to the RTTs involving local middleboxes, such as wireless gateways and VPN servers.

7. CONCLUSION AND FUTURE WORK

Using packet traces collected at three vantage points, we analyze several common techniques of passive RTT measurement from methodological perspective and examine several TCP characteristics and their effect on these techniques. Contrary to a common perception, we find significant presence of stretch acknowledgements (which can affect the accuracy of RTT estimation) and relatively low use of TCP timestamps (which reduces the applicability of some measurement techniques). We quantify the impact of regular delayed acknowledgements, as well as stretch acknowledgements, on RTT measurement accuracy, and find that while delayed acknowledgements sometimes inflate RTT estimation, and – depending on the nature of measurement – cannot be ignored, stretch acknowledgements do not distort RTT estimation. We further find that even with the impact of delayed ACKs factored out, the RTT measured in the course of a connection often exceed those during the handshake, and identify middleboxes to be responsible for this behavior. Finally, we show that local delays (within campus network) can contribute a sizable portion of the overall delays and should not be ignored.

While we considered implications of stretch acknowledgments on RTT measurement, their significant presence in TCP communication can have far-reaching implications for TCP performance. We hope future research will investigate this phenomenon in greater detail.

Acknowledgements

The authors are indebted to the LANDER project (John Heidemann of ISI and Christos Papadopoulos of Colorado State University and their teams), KC Claffy of CAIDA, and CWRU' ITS organization (especially Jim Nauer, Kevin Chen, Dan Matthews, and Roger Bielefeld) for their help in procuring data for this study. We are grateful to Mark Allman of ICSI for extensive comments on an early draft of this paper.

8. REFERENCES

[1] The CAIDA UCSD Statistical information for the CAIDA Anonymized Internet Traces. http://www.caida.org/data/passive/passive_trace_statistics.xml.

[2] University of Southern California-Information Sciences Institute dataset usc_lander_ongoing_tracing_scrambled-20120501. 01/16/2015. https://www.predict.org/.

[3] J. Aikat, J. Kaur, F. D. Smith, and K. Jeffay. Variability in TCP round-trip times. In *IMC*, 2003.

[4] M. Allman. A Web Server's View of the Transport Layer. *CCR*, 30(5):10–20, Oct. 2000.

[5] M. Allman, V. Paxson, and E. Blanton. TCP congestion control. RFC 5681, 2009.

[6] H. Balakrishnan, V. N. Padmanabhan, and R. H. Katz. The effects of asymmetry on TCP performance. *Mobile Networks and applications*, 4(3):219–241, 1999.

[7] H. Ding and M. Rabinovich. TCP stretch acknowledgements and timestamps: Findings and implications for passive RTT measurement. Extended paper. http://engr.case.edu/rabinovich_michael/otherPubs/rtt.pdf.

[8] G. Fairhurst, N. K. G. Samaraweera, M. Sooriyabandara, H. Harun, K. Hodson, and R. Donadio. Performance issues in asymmetric TCP service provision using broadband satellite. *Communications, IEE Proceedings-*, 148(2):95–99, Apr 2001.

[9] S. Floyd, A. Arcia, D. Ros, and J. Iyengar. Adding acknowledgement congestion control to TCP. RFC 5690, 2010.

[10] J. Gettys and K. Nichols. Bufferbloat: dark buffers in the Internet. *Communications of the ACM*, 55(1):57–65, 2012.

[11] J. Huang, F. Qian, Y. Guo, Y. Zhou, Q. Xu, Z. M. Mao, S. Sen, and O. Spatscheck. An in-depth study of LTE: effect of network protocol and application behavior on performance. In *ACM SIGCOMM*, 2013.

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

[13] H. Jiang and C. Dovrolis. Passive estimation of TCP round-trip times. *CCR*, 32(3):75–88, 2002.

[14] M. Kühlewind, S. Neuner, and B. Trammell. On the state of ECN and TCP options on the internet. In *PAM*, 2013.

[15] A. Medina, M. Allman, and S. Floyd. Measuring the evolution of transport protocols in the internet. *CCR*, 35(2):37–52, 2005.

[16] I. T. Ming-Chit, D. Jinsong, and W. Wang. Improving TCP performance over asymmetric networks. *CCR*, 30(3):45–54, 2000.

[17] V. E. Paxson. *Measurements and analysis of end-to-end Internet dynamics*. PhD thesis, UC, Berkeley, 1997.

[18] L. Qian and B. E. Carpenter. Some observations on individual TCP flows behavior in network traffic traces. In *11th Int. Symp. on Comm. and Inf. Technologies*, 2011.

[19] S. D. Strowes. Passively measuring TCP round-trip times. *CACM*, 56(10):57–64, 2013.

[20] B. Veal, K. Li, and D. Lowenthal. New methods for passive estimation of TCP round-trip times. In *PAM*, 2005.