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 a 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 affect the accuracy of RTT estimation. Finally, after removing possible errors that due to different passive RTT measurement technologies, we show unrealized potential for RTT improvement on the Internet.

Keywords

RTT measurement, TCP delayed and stretch acknowledgement, TCP timestamp

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 existing techniques of passive RTT measurement and examines several TCP characteristics and their effect on these techniques.

In particular, using a packet trace collected at the edge of a campus network, 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 [18], or over 92% of flows originating from customers of a particular LTE network operator carry them [9], we find their use in our actual traffic to be low (around 30% of flows). We further verify these findings on other two open traces. Limited timestamp usage affects the applicability of some measurement techniques as discussed later. At the same time, we find significant use of so called *stretch acknowledgements* [15], 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, 7, 14]), to our knowledge, ours is the first study to measure their actual usage on the Internet.

Prior work noted – but did not evaluate – the potential effect of delayed acknowledgements on accuracy of RTT estimation [3]. We quantify this effect and further show that even after removing the possibility of the regular delayed acknowledgement bias, the techniques relying on data segment acknowledgements tend to overestimate RTT. We further find that this may due to the low performance of wifi connections.

Finally, using passive RTT measurement techniques unaffected by the above biases, we consider RTT distribution of actual Internet communication from our three vantage points and show that RTTs of actual communication have barely improved since prior studies of more than 10 years ago, and the potential RTT improvement on the Internet remains unrealized. Furthermore, local RTT within campus intranet, which is sometimes neglected (e.g., [3, 16]), may constitute a substantial portion of the total network delay.

2. PASSIVE RTT MEASUREMENT

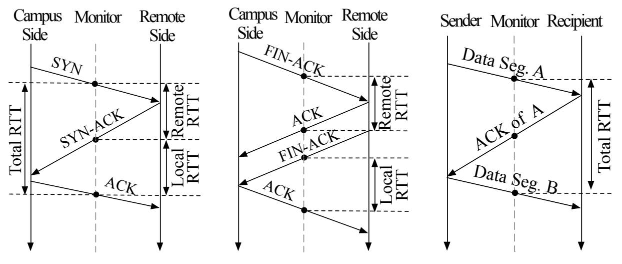
Passive RTT measurement techniques fall into two broad categories - those that require bidirectional packet trace and those that only need to see packets in one direction. Our setup allows us to capture bidirectional traffic and we mostly focus on the bidirectional techniques. Bidirectional techniques 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. Figure1a 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 similar method utilizes segment association during the closing of a TCP connection, to which we refer as *teardown* technique. Figure 1b 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. This method also applies to simultaneous close, but frequently abnormal TCP closing makes it 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

[*]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.



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

Figure 1: Passive RTT estimation methods

shown in Figure 1c, 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 [11] or congestion avoidance [10] phase. The TCP *timestamp* method [18] 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 [18]. It utilizes discrete autocorrelation function to detect the gaps between TCP bursts. However, the algorithm to detect the burst-gap patterns requires several pre-defined 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 (Cleveland) and Colorado State (Fort Collins, CO) [2], and the third is a TCP packet trace of the traffic on a backbone link in Chicago provided by CAIDA [1][1] All captures are done using Endace 10Gbps cards. We estimate losses in measurement instrumentation below. Because 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 CWRU and CSU traces are bidirectional, the CAIDA trace typically sees packets in only one direction in a given connection due to asymmetric routing. Thus further limits the questions we could answer using the CAIDA trace.

[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 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%

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%

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 flows and – in case multiple connections reuse the ports – further splitting each flow by locating TCP handshakes within each flow (which happens in only less than 2% of the flows 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. To estimate packet loss in the measurement instrumentation, 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 the 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. To avoid distorted RTT readout due to retransmitted or reordered packets, we do not use retransmitted data packets or any data packets between retransmitted 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 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. Table 3 shows fine grained OS distribution of CWRU Trace.

4. STRETCH ACKNOWLEDGEMENTS

TCP expects a recipient to acknowledge at least every other

Table 3: Fine-grained OS Distribution of flow sources (CWRU Trace)

	Flows	Win 7 or 8	Win Others	Mac 10.X/newer	Mac Others	Linux 2.X	Linux 3.X	Unkwn
Local Clients	6.8M	66%	4%	11%	7%	6%	4%	1%
Local Servers	2.3M	24%	2%	0	<0.1%	49%	9%	16%
Local Total	9.1M	55%	4%	8%	5%	17%	5%	5%
Remote Clients	2.3M	43%	11%	5%	3%	14%	20%	3%
Remote Severs	6.8M	4%	<1%	<0.1%	0	5%	44%	45%
Remote Total	9.1M	14%	3%	1%	1%	7%	38%	35%

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., [10]). 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 [15]). This finding is interesting because, although there were a number of proposals to use stretch ACKs in various asymmetric environments (e.g., [6,7,14]), RFC 5690 noted in 2010 that the authors were "not aware of the deployment of ACK filtering in the Internet" [8], and RFCs 2525 and 4413 both refer to stretch ACKs as a TCP implementation bug.

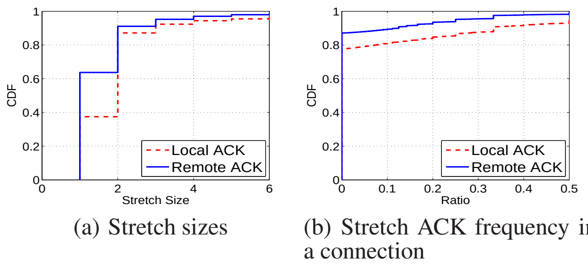
To measure the prevalence of stretch acknowledgements, for each connection we measure the length of the data packet trains (we call *stretches*) between consecutive (i.e., without an ACK with intervening acknowledgment numbers[2]) cumulative ACKs. Again, we exclude potential distortions by excluding 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.

4.1 Prevalence

Figure 2a shows the distribution of stretch sizes 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 2b 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% stretches from this flow are greater than 2. It can be seen from Figure 2b that 19% of all local flows, and 11% of all remote flows, used stretch ACKs.

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



(a) Stretch sizes (b) Stretch ACK frequency in a connection

Figure 2: Prevalence of stretch ACKs in CWRU trace

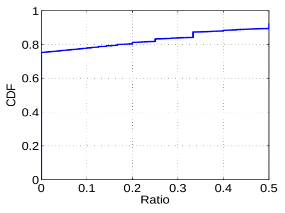


Figure 3: Per-connection stretch ACK frequency in CSU trace

Our overall conclusion is that stretch ACKs are a fairly common occurrence. 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 has been suggested as the means to reduce router transmission queues and improve TCP throughput in networks with asymmetric links [6, 14]. 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 4 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 all the flows (see Table 2). Table 5 shows fine grained OS Distribution of both local and remote flows with stretch ACKs, which can be similarly compared with Table 3. This speaks against the end-hosts being the culprit, leaving the network elements as a likely cause. Section 6 below provides further support to this conjecture.

Table 4 does show some increase in the prevalence of Win-

Table 4: 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%

Table 6: Timestamp Option Prevalence

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

dows 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.

5. TCP TIMESTAMPS

Previous studies that relied on TCP timestamps for passive measurements provided indirect indication of their common use: [18] and [12] note that, respectively, over 75% and 83% of the servers they probed supported the timestamp option, [9] report that over 92% of flows originating from customers of a particular LTE network carry them, and [17] 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 all three datasets. Table 6. 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 prevalence of timestamp use.

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 (916M connections), and find out that only 28% 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 7. It can be seen that Windows clients disproportionately contribute to the low prevalence of timestamp use.

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 in terms of the wide support of TCP timestamps among servers but not the implication that this indicates their wide use.

Table 7: 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%

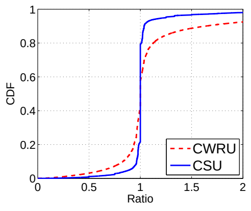


Figure 4: Teardown to handshake RTT ratio

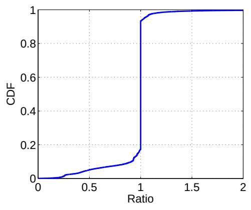


Figure 5: Delayed ACK effect on Data-ACK RTT

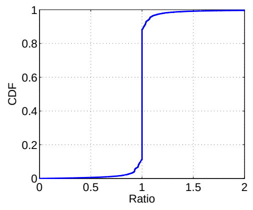


Figure 6: Stretch ACK effect on Data-ACK RTT

Table 8 shows the fine grained OS Distribution of both sources of flow in CWRU trace that do not use timestamps.

To our knowledge, the only study that passively measured TCP timestamp use is [13] 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). While the data we use shows growth in TCP timestamp use over the past 10 years, it 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.

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 a close that allows unambiguous teardown estimation. 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. For example, in the Data-ACK method, when a data segment arrives and the receiver

Table 5: Fine-grained OS distribution of sources of flows with stretch ACKs

	Flows	Win 7 or 8	Win Others	Mac 10.X/newer	Mac Others	Linux 2.X	Linux 3.X	Unkwn
Local Side	781K	65%	4%	9%	2%	9%	7%	3%
Remote Side	371K	15%	2%	3%	<1%	9%	30%	40%

Table 8: Fine-grained OS Distribution of flow sources without timestamp support (CWRU trace)

	Flows	Win 7 or 8	Win Others	Mac 10.X/newer	Mac Others	Linux 2.X	Linux 3.X	Unkwn
Local Clients	4.8M	92%	6%	<0.1%	<0.1%	<1%	<0.1%	1%
Remote Clients	1.3M	75%	20%	<0.1%	<0.1%	3%	<0.1%	1%
Local Servers	29K	1%	1%	0	0	29%	29%	40%
Remote Severs	306K	<1%	<1%	0	0	5%	25%	69%

has already acknowledged all the previously received data, the acknowledgement can be delayed by up to 500ms thus inflating the RTT estimate. 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. 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). 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 5 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 inflation from "normal" delayed ACKs quantified 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 eliminates RTT inflation from stretch ACKs if a host acknowledges at least every third data segment but not if the host allows longer stretches. Our assessment hence underestimates any inflation.

Figure 6 presents the CDF of the ratios of the median values of the above two kinds of sample RTTs for all 328K connections 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 ACKs observed in the traces, leaving in-network filtering as the likely reason.

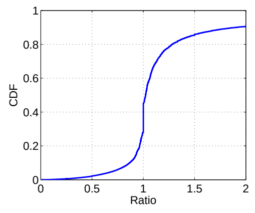


Figure 7: Timestamp to handshake RTT Ratio

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 (see Figure 4) 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, because a timestamp RTT sample can be distorted when the sender on Figure 1c does not immediately have data segment B to send, we exclude timestamp RTT samples over 2000ms.

To see if these filtered timestamp RTT measurements successfully mitigates distortion, we compare them with the data-ACK measurements. As handshake and timestamp methods allow us to generate both local and remote RTT samples, we calculate all local and remote handshake, timestamp and data-ack samples in each connections, and compared both the local/remote timestamp-to-handshake and data-ack-to-handshake ratios for 159K connections[3], as shown in Figure 8 and Figure 9. It can be seen there is virtually no difference between the two curves for both local and

[3]As we always eliminate the impact of normal delayed ACK in both timestamp and data-ack methods, these connections are all of the connections that could generate all kinds of RTT samples

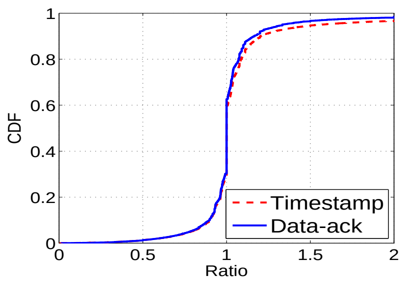


Figure 8: Remote Timestamp-to-handshake Ratio and Data-ack-to-handshake Ratio Comparison

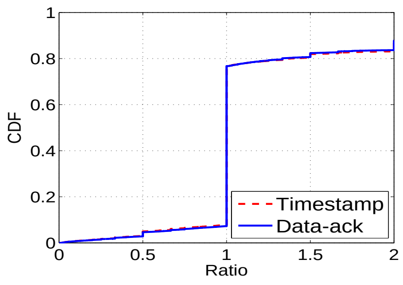


Figure 9: Local Timestamp-to-handshake Ratio and Data-ack-to-handshake Ratio Comparison

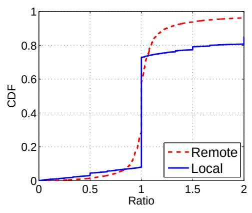


Figure 10: Timestamp to handshake RTT ratios for local and remote RTTs

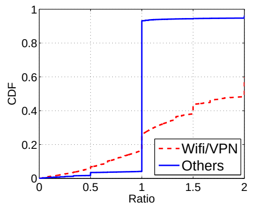


Figure 11: Timestamp to handshake local RTT ratios per connection types

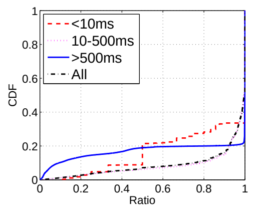


Figure 12: Remote-to-total RTT ratios using handshake estimates

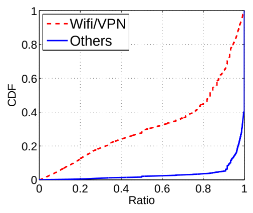


Figure 13: Remote to total RTT ratios for Wifi/VPN hosts and other hosts

remote RTTs, which means that the data-ack and filtered timestamp estimates match closely.

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 7 plots the CDF of these ratios. 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 are 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 attempts to shed light on the reasons behind this phenomenon.

Our first question is whether the inflation is due to local or remote delays. Since segment exchanges in both timestamp and handshake methods allow one-sided RTT measurements, Figure 10 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% 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 11 presents the corresponding CDFs of the RTT ratios. Indeed, it shows that virtually all the skew observed in local RTTs in Figure 11 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 (within campus network, using CWRU trace only) to the overall delays, which is sometimes neglected in RTT measurements (e.g., [3, 16]). 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 12 plots CDFs of the ratios of remote to total RTTs for 9.1M connections with handshake samples. According to Figure 12, for about 12% of all samples, the remote RTT accounts for no more than 80% of the total RTT. Further, this finding 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 (over 25%

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 so high in the first place due to local RTT, presumably some hiccup within the intranet (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 13 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.

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

We analyze a packet trace collected at the edge of campus network, along with other two open traces, from the perspective of extracting passive RTT measurements. Our key findings are as follows. Contrary to a common perception, we find relatively low use of TCP timestamps (which reduces the applicability of some measurement techniques) and significant presence of stretch acknowledgements (which can affect the accuracy of RTT estimation). We quantify the impact of regular delayed acknowledgements on RTT measurement accuracy, and show that even with their impact factored out, the RTT measurements in the course of a connection often exceed those during the handshake. We show that local delays (within campus network) can be responsible for a significant part of the overall delays and should not be ignored. Finally, we show that RTTs of actual communication could vary a lot, and the potential RTT improvement on the Internet remains unrealized.

The significant presence of stretch acknowledgements could have multiple impact on the TCP protocol. The lack of acknowledgements could slower the phase of slow start. As it might also add the waiting time for the sender, it might cause more packet retransmission. Besides, it could invalidate existing research proposals(like RFC 5690), and affect precision of RTT measurement as we discussed. As stretch acknowledgement is proposed as an optimization to the current network, to see how much traffic it saved and the consequences on the actual network could be our future work.

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 Bielfeld) for their help in procuring data for this study.

Further Results

This section presents additional observations from the CWRU trace regarding RTT estimation.

7.1 Stretch ACKs in Busy and Quiet Times

Since some network elements are known to filter ACKs , we inspect the flows from known middleboxes in the CWRU network, namely, all campus wireless gateways, which we were able to identify by obtaining the list of their external IP addresses. We collected two additional traces of traffic flowing through these wi-fi routers

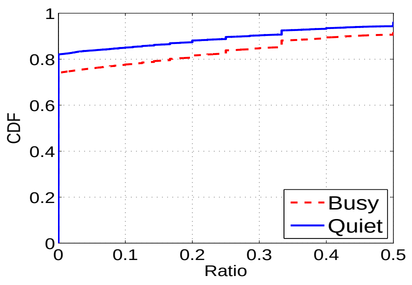


Figure 14: Stretch ACK frequency in flows of local wifi hosts

at a busy and quiet time[4] and compare the prevalence of stretch ack connections in both traces. If the wireless gateways are responsible for ack filtering, the prevalence of stretch ACKs would be higher during the busier time of the day, when it is more likely that a newly arriving ACK would find an earlier ACK already in the queue that could be dropped. Figure 14 shows the fraction of greater-than-2 stretches of the two traces for 623K (busy) and 94K (quiet) local flows. The fraction of local flows with stretch ACKs (recall that these are flows with at least 10% of stretches longer than 2) drops from 22% to 15% from the busy to quiet period. We also used the p0f tool to consider the distribution of the operating systems of the clients behind the gateways. Table 10 shows the OS distribution of the all local flow sources behind the wireless gateways using stretch ACKs, and Table 9 shows the OS distribution from all local flow sources behind the wireless. For ease of comparison, Table 11 shows the OS distribution of non-stretched local flows of the busier/quieter wifi period. (This table could be produced by arithmetic calculations over the results in the other two tables.)

These results are inconclusive. We do see some increase in stretch ACKs during busy period. However, we also see an increase in Windows prevalence compared to the general trace, which as mentioned can be configured to use stretch ACKs. At the same time, when looked at the OS distribution difference between the busy and quiet traces, we see that the prevalence of stretch ACKs in the busy period increases while the prevalence of Windows OS if anything *decreases* slightly. We also ran the same test on two other busy-quiet periods in August 2014. That experiment also showed an increase in stretch ACKs during busy time, from 9% to 17%. In general, these experiments provide some additional weak evidence in support of ACK filtering in the network.

7.2 Remote RTT Inflation of Local WiFi Hosts

Figure 15 plots timestamp to handshake remote RTT ratio for wifi hosts and other hosts respectively. For the wifi curve there are 81K samples and in 35% cases tiemstamp RTT are greater than handshake, in 18% cases lower. For other hosts, there are 268K samples and in 43% cases timestamp are higher, 31% timestamp are lower.

Figure 16 plots data-ack to handshake remote RTT ratio for wifi hosts and other hosts respectively. For the wifi curve there are 233K samples and in 34% cases data-ack RTT are greater than handshake, in 23% cases lower. For other hosts there are 1.28M samples and in 36% cases data-ack are higher, 36% data-ack are lower.

[4]Busy period, comprising 349.2M TCP packets in 1.3M valid connections, and quiet period, comprising 76.9M TCP packets in 250K valid connections, respectively.

Table 9: OS distribution of sources of all local wifi flows

	Flows	Win 7 or 8	Win Others	Mac 10.X/newer	Mac Others	Linux 2.X	Linux 3.X	Unknown
Busier Period	1.2M	40%	4%	17%	26%	1%	11%	<0.1%
Quieter Period	237K	37%	3%	14%	29%	1%	15%	<0.1%

Table 10: OS distribution of sources of local wifi flows with at least 10% stretch ACKs

	Flows	Win 7 or 8	Win Others	Mac 10.X/newer	Mac Others	Linux 2.X	Linux 3.X	Unknown
Busier Period	136K	56%	7%	10%	19%	1%	7%	<0.1%
Quieter Period	14K	47%	9%	9%	23%	1%	10%	<0.1%

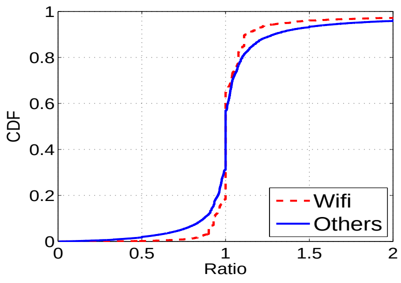


Figure 15: Timestamp to handshake Remote RTT ratio for Wifi hosts and Other hosts

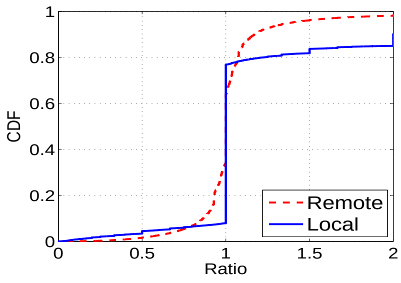


Figure 17: Data-ack to handshake remote/local RTT ratio

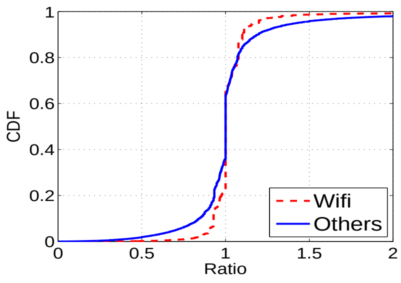


Figure 16: Data-ack to handshake Remote RTT ratio for Wifi hosts and Other hosts

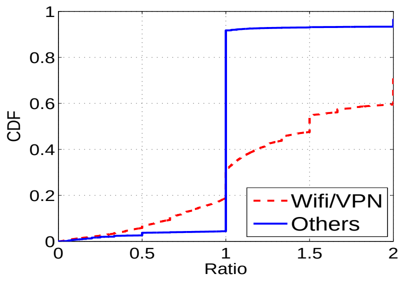


Figure 18: Data-ack to handshake Local RTT ratio for Wifi/VPN hosts and Other hosts

These results show that the flows allowing timestamp RTT estimation still shows some remote RTT inflation while a much larger set of flows that allow data-ack estimations show virtually no remote RTT inflation. This is understandable because as we saw, flow with timestamps are sent disproportionally by non-windows systems, and these systems are disproportionally represented among devices using wifi (as these are often smartphones that use Android or IOS). The difference in the OS mix among hosts is highlighted in Table 12, which shows the fine grained OS distribution of local clients behind wifi/VPN middle-boxes of the CWRU trace.

Curiously, we also see that *remote* RTTs of flows with *local* hosts behind wifi gateways show some inflation compared to other hosts. We have no explanation for this observation except to speculate that servers serving mobile apps exhibit different behavior.

7.3 Other RTT results

Figure 17 plots data-ack to handshake ratio for local and remote RTT respectively. For the local RTT there are 2.28M samples and in 23% cases data-ack RTT are greater than handshake, in 8% cases lower. For remote RTT there are 1.5M samples and in 36% cases data-ack are higher, 34% data-ack are lower.

Figure 18 plots data-ack to handshake ratio for local wifi/VPN hosts and other hosts respectively. For the wifi/VPN curve there are 562K samples and in 68% cases data-ack RTT are greater than handshake, in 19% cases lower. For other hosts there are 1.7M samples and in 8% cases data-ack are higher, 4% data-ack are lower.

Figure 19 plots the comparison of data-ack to handshake Local RTT ratio for Wifi and VPN hosts at beginning and throughout the connection respectively(both curves 401K samples from the same connections). The inflation at the beginning of the connection (in 48% cases data-ack RTT is greater, 29% data-ack is smaller than handshake) is much smaller than thoughout the connection (in 67% cases data-ack is greater, 19% data-ack is smaller), which suggests that local queueing delay in the network middle-box is the main reason for these local RTT inflation.

Table 11: OS distribution of local Wifi clients with flows without stretch ACKs

	Flows	Win 7 or 8	Win Others	Mac 10.X/newer	Mac Others	Linux 2.X	Linux 3.X	Unkwn
Busier Period	1.1M	38%	5%	18%	27%	9%	2%	<0.1%
Quieter Period	223K	36%	3%	14%	30%	1%	15%	<0.1%

Table 12: OS Distribution of local clients behind wifi/VPN middle-boxes

	Flows	Win 7 or 8	Win Others	Mac 10.X/newer	Mac Others	Linux 2.X	Linux 3.X	Unkwn
Local Wifi Clients	1.2M	44%	3%	22%	20%	1%	9%	<1%
Local VPN Clients	458K	57%	6%	22%	14%	<0.1%	<0.1%	1%
Local Wifi Clients without timestamp	582K	92%	7%	<0.1%	<0.1%	<0.1%	<0.1%	<1%
Local VPN Clients without timestamp	288K	90%	10%	0	0	0	0	<1%

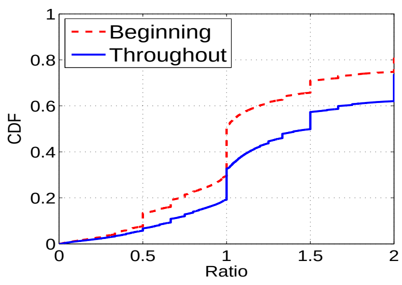


Figure 19: Data-ack to handshake local RTT ratio for Wifi/VPN hosts at beginning and throughout the connection

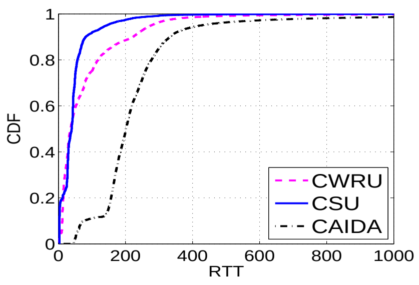


Figure 21: Per-connection Distribution of Handshake RTT Estimates

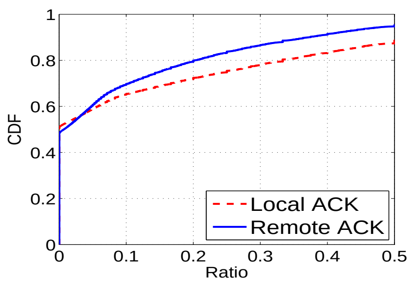


Figure 20: Stretch ACK frequency in longer connections

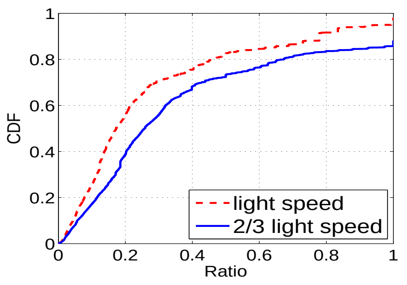


Figure 22: Per-flow propagation-delay-to-RTT ratio

7.4 Stretch ACK prevelance of longer connection

If the filtering on network to be responsible for the cause of stretch ack, then it is understandable that flows with more data packets are more likely to give opportunity for the other side to use stretch ack. To see if this is the truth, we consider flows with more than 20 data packets, and plot the distribution of fractions of greater-than-2 stretches (out of all stretches) in each direction if we see at least 20 data packets from the other side. As show in Figure 20, we in total inspect 997K local flows and 357K remote flows, and it shows that 35% local and 31% remote relatively long flows are using stretch ack.

7.5 Actual RTT distribution and Propagation delay

We turn to actual network delays exhibited in all campus communications and backbone links. As handshake estimate is applicable to both bidirectional and one-direction flows, and could generate total RTTs from most connections, Figure 21 plots the CDF of RTT estimates using handshake methods for all the three traces, representing 9.1M(CWRU), 1.9M(CSU) and 8.7M(CAIDA) connections respectively. It can be seen that for all traces, mroe than 95% connections experienced a handshake RTT of less than 500ms. While the host from campus experience a much smaller RTT than backbone links, there is also a significant difference between different campuses. This highlights the difficulty in RTT comparison.

To see if the above finding is due to inherently remote com-

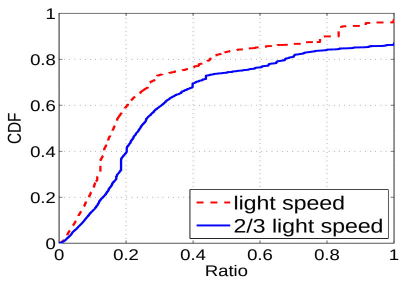


Figure 23: Per-host-pair propagation-delay-to-RTT ratio

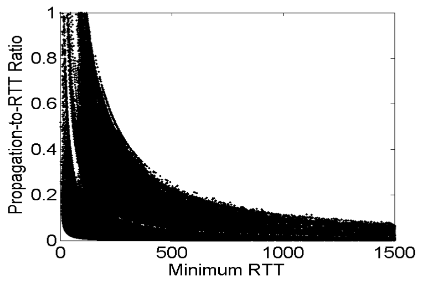


Figure 24: Per-flow Relationship between propagation-delay-to-RTT ratio and minimum RTT

munication, we consider *routing stretch*, i.e., the difference of our RTTs from the theoretical minimum as represented by the straight-line speed-of-light delay between the geographical locations of the end-points (determined from IP addresses using the Maxmind GeoIP database). As the the propagation speed depends on the physical medium of the link and could drop to 2/3 of the initial value, we consider both boundary propagation speeds in the study. Figure 22 and Figure 23 show the ratio of ideal-RTT-to-observed-RTT ratio per flow (8.9M samples) and per host-pair(1.2M samples) respectively by using the minimum RTT value observed in each flow and host-pair of our campus network, i.e. CWRU trace. In all cases, for more than 70% of all the samples physical propagation only accounts for less than half the overall delay, which offers plenty of untapped opportunity for network optimization[5]. We also plots the relationship between the propagation-to-RTT-delay ratio and actual RTT. Figure 24 plots the distribution of per-connection propagation-delay-to-RTT ratio and the correspondence minimum RTT measured from that connection using 2/3 of the light speed.

8. REFERENCES

[1] The caida ucsd statistical information for the caida anonymized internet traces. http://www.caida.org/data/passive/passive_trace_statistics.xml. Accessed: 2015-01-20.

[2] Usc/isi ant datasets and dataset formats. http://www.isi.edu/ant/traces/. Accessed: 2015-01-20.

[3] J. Aikat, J. Kaur, F. D. Smith, and K. Jeffay. Variability in TCP round-trip times. In *3rd ACM SIGCOMM Conf. on Internet measurement*, pages 279–284, 2003.

[4] M. Allman. A Web Server's View of the Transport Layer. *Computer Communications Review*, 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] 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.

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

[9] 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 Conf.*, pages 363–374, 2013.

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

[11] H. Jiang and C. Dovrolis. Passive estimation of TCP round-trip times. *ACM SIGCOMM Computer Communication Review*, 32(3):75–88, 2002.

[12] M. Kühlewind, S. Neuner, and B. Trammell. On the state of ECN and TCP options on the internet. In *Passive and Active Measurement*, pages 135–144. Springer, 2013.

[13] A. Medina, M. Allman, and S. Floyd. Measuring the evolution of transport protocols in the internet. *ACM SIGCOMM Computer Communication Review*, 35(2):37–52, 2005.

[14] I. T. Ming-Chit, D. Jinsong, and W. Wang. Improving TCP performance over asymmetric networks. *ACM SIGCOMM Computer Communication Review*, 30(3):45–54, 2000.

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

[16] 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 (ISCIT)*, 2011.

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

[18] B. Veal, K. Li, and D. Lowenthal. New methods for passive estimation of tcp round-trip times. In *Passive and Active Network Measurement*, pages 121–134, 2005.

[5]There are exception ratios (greater than 1) in Figure 22 and Figure 23, which we attribute to network NATs and inaccuracies in the GeoIP database.