A Simple Passive Method to Estimate RTT in High Bandwidth-Delay Networks

Iria Prieto,
Mikel Izal,
Eduardo Magaña
and Daniel Morato

Public University of Navarre
Navarre, Spain
Email: iria.prieto, mikel.izal, eduardo.magana, daniel.morato @unavarra.com

***Abstract*—This paper presents a simple passive algorithm to estimate the Round-Trip-Time (RTT) of a TCP connection in high bandwidth-delay network scenarios. In these scenarios, a passive RTT estimator that can be used on captured packet traces is a useful tool for performance analysis. The algorithm is based on observing periodic RTT patterns in a TCP connection that is not filling its $bandwidth \times delay$ product. The results are compared to other passive methods such as the RTT of initial TCP handshake or TCP Timestamp option samples. The algorithm is shown to be effective and may be used in more scenarios than other methods thus, it provides a valuable tool to improve the amount of TCP connection whose RTT can be measured in a captured packet trace.**

Keywords–RTT; passive; network; traffic.

I. Introduction

Round-Trip-Time (RTT) is a key network performance metric. It is easy to measure using active probes (i.e., with Internet Control Message Protocol, ICMP) but it is not so simple to estimate by passive observation of traffic. There are situations where a passive RTT estimation from a captured packet trace is needed in the field of network and system analysis and network health check.

The RTT indicates the time that is taken to obtain a response from the other side of the communication, namely, the network latency. The latency can be used to identify short-life network problems. For instance, a RTT instantaneous peak may indicate a short period of congestion or suggest the existence of a network problem. Furthermore, the RTT of different network segments can be used to analyze if a performance issue is due to different parts of the network or even suggest it may be an application or server issue. RTT is a major factor in TCP, Transmission Control Protocol, connection and data transfer performance.

RTT is calculated as the elapsed time between one packet sent by one endpoint and the reception of a packet from the other endpoint that acknowledges the first packet. Any packet that can be guaranteed to have been sent by the other endpoint only after reception of the first packet may be used. There are several factors that can affect the measured time, like retransmissions or packet losses. Also, the other side may delay the response for protocol reasons (like TCP delayed ACK) or just because the response is not mandatory and the data flow in the other direction is being used as response. The later case can be defined as limited by the application.

All these factors make the passive estimating of RTT a non trivial task as it was exposed by Zhang [1]. It requires to take into account several conditions: disorders, retransmissions, lost packets, where the capture is located, etc.

Some RTT passive estimation methods have been proposed in the literature. The work of Strowes [2] is based on the observation of packets using TCP header timestamp option which is used by TCP protocol to generate RTT measures. For this samples to be present in a TCP connection the option TCP Timestamp option has to be activated, [3]. A TCP connection uses TCP options if both endpoints agree to use it at connection establishment negotiation. Another work which used Timestamp for RTT estimation is [4]. Both works show that the estimation solved some passive approach problems such as packet loss or capture point dependence. They also showed that Timestamp estimation is as good as the use of active ICMP probes. The main problem of TCP timestamp method is that there is still a very large fraction of TCP connections that do not use Timestamp option. As an example, our measurements at an university access link show only 22% of observed TCP connections successfully negotiated the use of TCP timestamp option. The test was performed on a trace of 1000 TCP connections captured during one work-day on Nov 2014.

Other passive methods work regardless of TCP timestamp option being used. For instance, the authors in [5] estimated the RTT value through the three-way handshake and the slow-start phase. The RTT provided by the three-way-handshake is not always accurate because it may be changed by middleboxes between client and server. These middleboxes may answer or initiate the connection on their own, resulting in lower RTTs. Besides, extracting parameters from time measures of TCP slow start phase is not an easy task. Other techniques relay on more complex mechanics. The authors in [6] associate a data segment with the ACK segment that triggered it. Other approaches try to measure RTT by mimicking changes in the sender's congestion window size [7]. It should be taken into account that these estimations are affected by packet losses, the TCP window scaling option and buggy TCP implementations.

The motivation for this work comes from the field of performance analysis of networks by means of captured packet traces. This is a valuable tool for troubleshooting and problem detection of large enterprise networks where packet traffic is captured at a vantage point to study network problems. This analysis is usually done by capturing traffic in advance and analyzing it a-posteriori in case some problem was reported. Thus, it is unfeasible to perform active measures of RTT. The large amount of traffic usually captured would make very difficult deciding, which endpoints to measure RTT in between. That is the reason why a passive RTT estimation tool is searched that can provide RTT values between the endpoints of every observed TCP connection. The scenarios of interest are high speed networks with middle to high traffic loads like those of datacenter or large enterprises.

The paper is organized as follows. First of all, the algorithm and configuration parameters are introduced. Section III describes the network scenario used to check the proposed algorithm. Section IV and V present the results and conclusions.

II. Proposed algorithms

The proposed algorithm provides an estimation of RTT by observing the behavior of TCP connections that are not filling its $bandwidth \times delay$ product. Note that a TCP connection data flow is limited by the flow control window which is advertised from each endpoint to the other.

Assuming an application, which always has data to send over TCP, if the RTT is high enough, TCP will be able to send the full permitted window of data and stop sending till it receives the confirmation for the first packet in the window. In that case data will be sent like and ON-OFF source with RTT period. If RTT is not so high, acknowledged packets will start arriving before the end of the window, resulting in more or less continuous data flow. The proposed RTT measuring method examines the TCP connection and infers the RTT from the observed ON-OFF pattern.

The idea is to provide an RTT estimation method for TCP connections requiring only passive capture of traffic. Even it may work just on some TCP connections depending on their advertised window and $bandwidht \times delay$ product, there are common scenarios where TCP window is small enough.

The algorithm evaluates if a given candidate RTT value could be the actual RTT seen by the TCP connection or it is an impossible value. The candidate is tested by using it as the time length interval to divide the connection time into slots and perform a simple check. If in any of the time slots more bytes have been sent by one endpoint than the advertised window, then the candidate is discarded. In fact, if a candidate is discarded it is clear that the actual RTT is lower than the candidate. If the candidate value passes the check it is an acceptable value for RTT. The algorithm searches for the smallest possible candidate value that can not be discarded as a valid RTT.

Defining the parameters,

- c: the total time that a connection lasts.
- t: the candidate time to be tested as possible value for the RTT.
- n: the number of t duration intervals on the connection $n = \lceil \frac{c}{t} \rceil$
- i: an interval being evaluated, $i \in 0..n$.
- B_i: Total bytes sent by the server in an interval, i.
- w_i: Maximum advertised window seen at interval i
- w_i^{max}: Maximum advertised windows seen up till interval i, $w_i^{max} = max\{w_k\} \quad \forall k \in 0..i$

The proposed algorithm consists in searching the smallest possible candidate t that does not fail the test. Several versions of the algorithm have been studied with different degrees of requirements. Depending on the test conditions different RTT estimators are generated named as RTT1, RTT2, RTT3.

1) RTT1: Candidate t is valid if in every interval, i, the total bytes sent is lower than the maximum window seen for each interval, equation 1
2) RTT2: Candidate t is valid if in every interval, i, the total bytes sent is lower than the maximum window seen for the whole TCP connection (discarding advertised window from TCP 3 way handshake packets SYN,SYN+ACK,ACK), equation 2
3) RTT3: Candidate t is valid if in every interval, i, the total bytes sent is lower than the maximum window seen up till that time in the TCP connection, equation 3

$$RTT1 = t \quad \forall i \in 0..n \quad B_i < w_i \tag{1}$$

$$RTT2 = t \quad \forall i \in 0..n \quad B_i < w_n^{max} \tag{2}$$

$$RTT3 = t \quad \forall i \in 0..n \quad B_i < w_i^{max} \tag{3}$$

In this work, the search for the smallest valid t is performed by initially choosing a candidate t that is clearly larger than the RTT and reducing t in a fixed amount δt every time the test is passed. When a value $t - \delta t$ fails the test, the previous t is declared the RTT estimation. The δt value used imposes the resolution of the estimator.

The algorithm is considered to finish when the candidate time fails the test. The value that will be shown as the final result is the previous one. The result of the algorithm is an upper limit from the theoretical RTT. In case that the first candidate time will not fail the test in the first iteration, it will not give any information since the real RTT could be higher than the tested one. In these cases, another candidate time could be chosen multiplying the first one by some value and the test would be restarted.

III. Validation scenario

The algorithm has been validated in the scenario of Figure 1. An emulated network of virtual machines is built with several client boxes in an emulated 10Mbps Ethernet. This virtual LAN, Local Area Network, is connected to a second virtual LAN through a routing virtual machine. In the second LAN there is a machine running a web server.

In the routing machine, the delay of packet forwarding is controlled using Netem tool [8]. The line speed of both virtual networks is 10Mbps.

The scenario is built with VirtualBox running on an Ubuntu 14.04 Linux PC. Client and router boxes are virtual boxes running Ubuntu 12.04. The host machine running the virtualization software acts also as the web server machine, Figure 1.

The scenario is configured for different RTTs by selecting the routing box forwarding delay, half RTT for each direction. The scenarios are configured for using total forwarding of

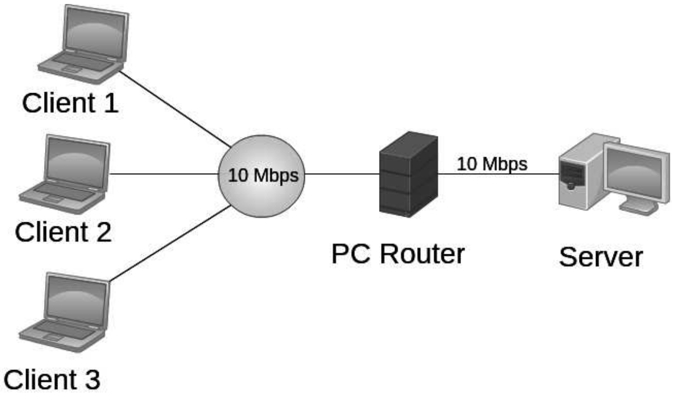

Figure 1. Emulated scenario of a network whose connections are limited by w/RTT.

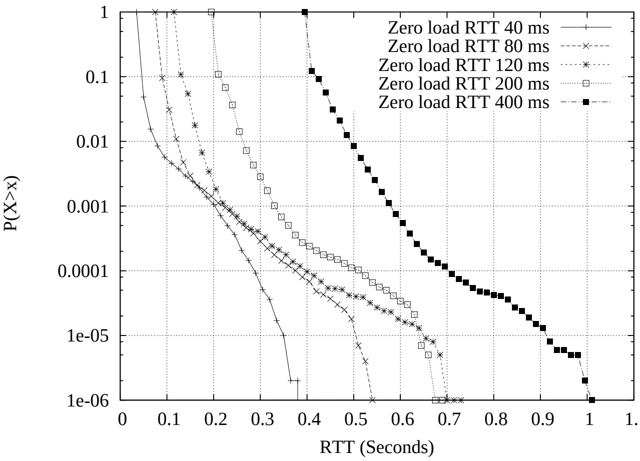

Figure 2. RTT estimation using the “Timestamp” method

40, 80, 120, 200 and 400 ms in the experiments. In order to emulate the behavior of middleboxes, which usually are used in high-performance networks, TCP handshake packets (first 3 packets with SYN, SYN+ACK and ACK flags) will have a lower forwarding delay, exactly a 10% less than the value used for the rest of the connection.

In order to have a scenario of TCP not filling the path $bandwidth \times delay$, TCP window scale option is deactivated.

The experiments consist on clients making HTTP, Hypertext Transfer Protocol, requests to the Web server. The server will send a variable size page whose size follows a uniform distribution between 1 and 3 MBytes. Clients will make requests with inter arrival times following a uniform distribution with mean 8 seconds. These characteristics will provoke that the channel of the server will have an average load about 6 Mbps.

IV. Results

The RTT of a path used by a TCP connection can be defined as the time it takes for a packet to travel from one endpoint to the other plus the time it takes the confirmation packet to return back to the original endpoint. That is a property of the path which could be calculated by just adding the link delays of the path. But an actual TCP connection is affected by the actual RTT of every packet it sends which is not always the pure RTT of the path because of variations due to waiting at queues along the path or response waiting times at the remote endpoint. Thus, the RTT can be seen as a random process. The estimation algorithms are trying to measure this random variable.

In the presented secenario, different RTT estimators have been evaluated, namely the three proposed RTT1, RTT2, RTT3 estimators provide a value for the RTT seen by a given TCP connection. They have been compared to two classical estimators of RTT for TCP connections: *initial RTT* estimator and *TCP timestamp option* estimator. Initial RTT estimator measures the RTT for the connection as the time from first SYN packet of the conception to the confirmation packet of the SYN+ACK packet. This is the time duration of TCP 3way handshake. The TCP timestamp estimator measures RTT of the connection by observing TCP timestamp options that provide accurate instant RTT samples. These samples are always greater than the actual RTT. The TCP timestamp estimator chooses the smallest sample value observed as the RTT for that connection path. The TCP timestamp option estimator will be a very good estimator by definition but its use depends on the captured connections using it.

Apart from that, as every TCP connection in the validation scenario has the same path, the full set of TCP timestamp measure samples could be used as a ground truth of the RTT random process. Figure 2 shows the probability density function of RTT for the different configurations.

To compare the estimators, 2000 connections were captured running the explained scenario. The five estimators RTT1, RTT2, RTT3, initial and TCP timestamp, were computed for every TCP connection seen.

The results, the mean, minimum, maximum and variance values, for all the estimators analyzed are shown in Table I. The probability density functions are shown in Figure 3. The results show that all estimators slightly overestimate the actual RTT and that the TCP timestamp is the most precise as expected. The overestimation is larger when the actual RTT to estimate has low values.

Analyzing the results obtained for the rest of the RTT estimators versus the initial RTT, obtained from the handshake, and the amount of time of the Timestamp, it is shown as the majority of the connections had results around the expected RTT, Figure 3. From the three algorithms, RTT1 is the one which overestimate less since it is the less strict.

Since the scenario emulates a network using middleboxes, the initial delay time which was obtained from the 3way handshake should be slightly lower than the delay for the rest of the connection, (10% lower). However, as the server experienced a moderate load the actual initial RTT was usually on the order of the RTT for the rest of the packets. This effect can be observed at the minimum value obtained for the initial RTT for each experiment, as shown in Table I.

Some connections had high values for all the methods except for the Timestamp estimation. During the connection, sometimes the actual packet RTT was the configured scenario value. In a long connection, at least some window flights experienced an RTT larger than the base one, due to the

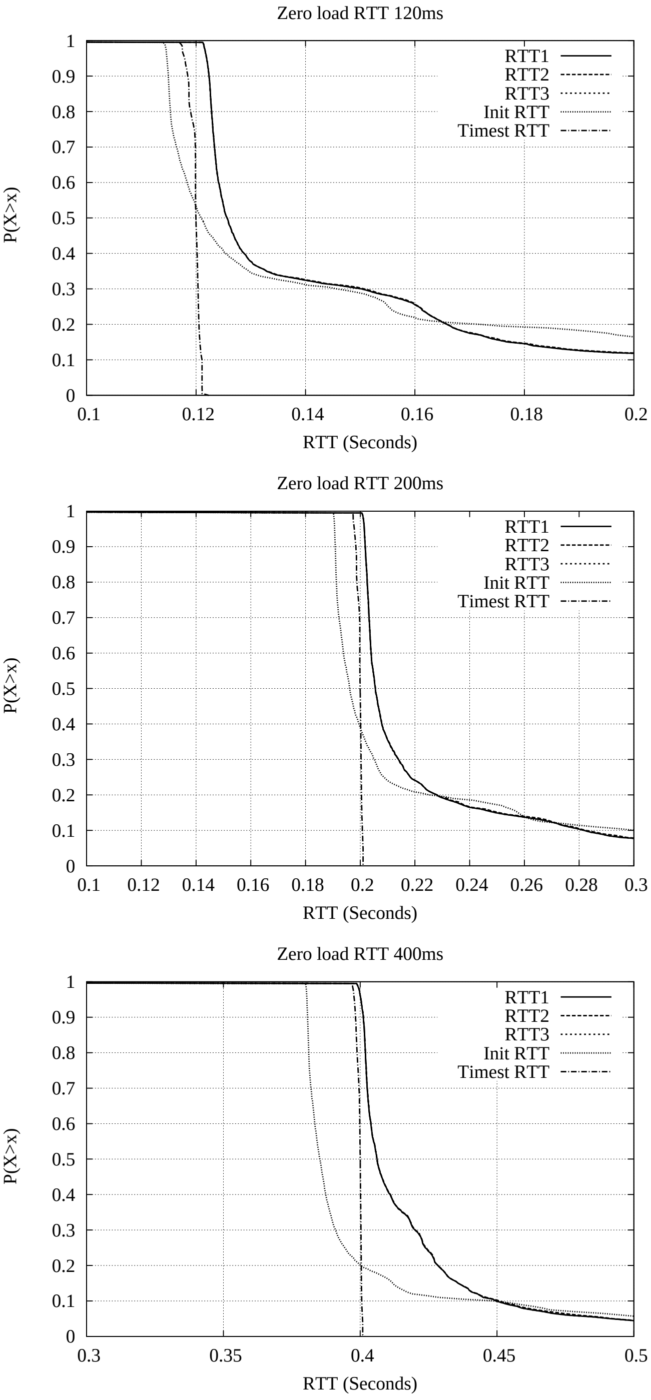

Figure 3. Comparing the RTT estimation using all methods

server being busy with traffic for others requests. The proposed algorithms adapt to the maximum time for the connection and so the values were higher than expected. Figure 4 shows an example for a connection whose RTT values were higher than expected. This connection should have a 400ms estimation for the RTT, according to the theoretical value, however the result of the algorithm for that connection was 700ms. The values obtained were similar to the maximum value for the Timestamp estimation observed for the same connection.

TABLE I. STATISTICS FOR THE CALCULATED RTT

Zero load RTT	Method	Min (s)	Max (s)	Mean (s)	Variance (s^2)
40ms	rtt1	0.054	0.396	0.087	0.003
	rtt2	0.054	0.396	0.087	0.003
	rtt3	0.054	0.396	0.087	0.003
	Initial	0.038	0.417	0.085	0.005
	Timestamp	0.036	0.041	0.040	0.000
80ms	rtt1	0.081	0.569	0.123	0.004
	rtt2	0.081	0.569	0.124	0.004
	rtt3	0.081	0.569	0.123	0.004
	Initial	0.074	0.569	0.126	0.006
	Timestamp	0.076	0.081	0.080	0.000
120ms	rtt1	0.121	0.660	0.151	0.003
	rtt2	0.121	0.660	0.151	0.003
	rtt3	0.121	0.660	0.151	0.003
	Initial	0.113	0.700	0.153	0.005
	Timestamp	0.115	0.122	0.120	0.000
200ms	rtt1	0.200	0.661	0.227	0.003
	rtt2	0.200	0.661	0.228	0.003
	rtt3	0.200	0.661	0.227	0.003
	Initial	0.190	0.712	0.224	0.005
	Timestamp	0.195	0.201	0.200	0.000
400ms	rtt1	0.398	0.945	0.422	0.002
	rtt2	0.398	0.945	0.423	0.002
	rtt3	0.398	0.945	0.422	0.002
	Initial	0.380	0.926	0.404	0.004
	Timestamp	0.396	0.402	0.400	0.000

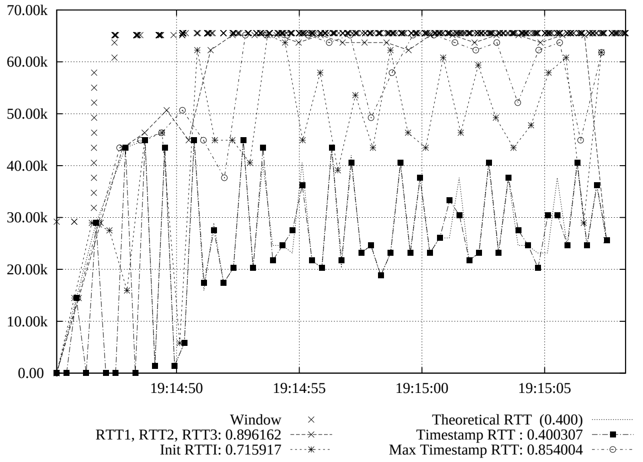

Figure 4. Observed timeseries of bytes for each RTT candidates.

Figure 5 shows the results from the estimators. The mean is slightly lower for the initial RTT compared to the proposed methods. The standard deviation is higher, which indicates a higher variability on the measurement. Besides, it is worth mentioning that for some cases the value obtained from the initial RTT is a subestimation of the real RTT.

Finally, it should be noted that the proposed algorithms check the validity of an RTT candidate by comparing the observed bytes to the advertised window. Thus it needs that the TCP connection is not filling the $bandwidth \times delay$ product for the path. Otherwise the initial candidate RTT is an upper RTT limit, giving no information. The limit is shown in Figure 6 where the minimum RTT for a given bandwidth is plotted. The points $(bandwidth, RTT)$ under the line represent situations where the algorithm does not work. This limit depends on the maximum advertised window allowed by TCP which is 64Kbytes for classical TCP but can be extended if it accepts the window scale option. Figure 6 shows this limit

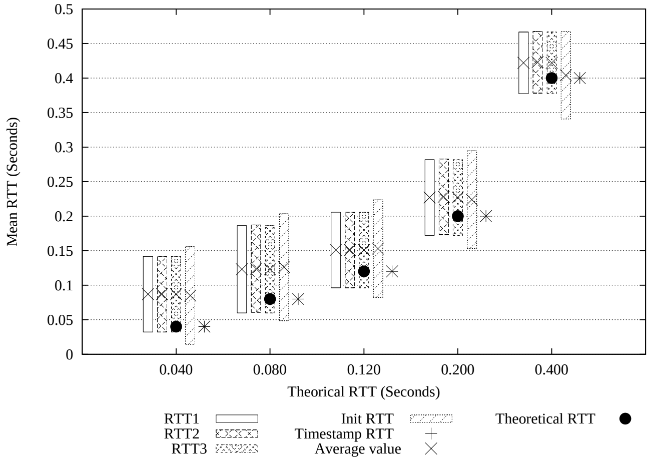

Figure 5. Average RTT and its deviation error obtained for all RTT tests.

for several values of the window scale option, from 64KB to 8MB.

For low bandwidth scenarios, the algorithm can be used to estimate the RTT, provided TCP connections do not use window scaling, or use low values. For higher speed scenarios TCP connections with larger window scale option values can be estimated. For example in a 1Gbps, data center network using window size of 2MB, it is possible to estimate an RTT larger than 16ms. In a longer link with 100Mbps, it is possible to estimate RTTs larger than 40ms provided the window size is 512KB or less. The low bandwidth (10Mbps) scenario used for validation without window scale option allows to estimate RTTs higher than 51.2ms. This can be checked at Table I where the result for RTT1, RTT2 and RTT3 in the 40ms scenario never gives an output lower than 52ms.

Nevertheless even if TCP endpoints agree to use certain window scale value, it does not imply they will advertise the maximum allowed window. Observations at authors' university access link show that even TCP connections usually negotiate window scales allowing up to 8MB advertised windows. However these connections afterwards do not advertise so large windows. Figure 7 shows the survival function of the maximum advertised window used by connections compared to the survival function of the maximum allowed advertised window for the negotiated window scale. Note that around 30% connections negotiate window scales that allow 512KB windows but approximately just 15% actually advertise 512KB. Thus around 85% of the observed link TCP connections will meet the requirements to estimate RTTs larger than 40ms.

V. Conclusions

The RTT value is a key metric for performance evaluation of a TCP connection. It determines the QoS perceived by the user especially in application level protocols with multiple requests like HTTP. The measurement of RTT is also a requisite for deeper analysis of TCP behavior from passive traffic captures.

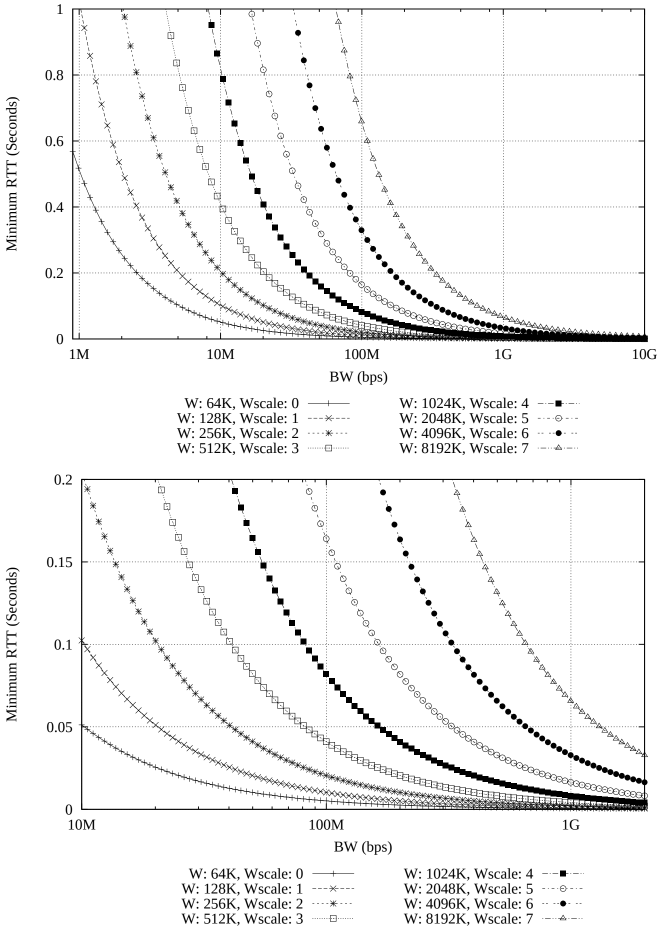

Figure 6. Mean RTT and its deviation error obtained for all RTT tests.

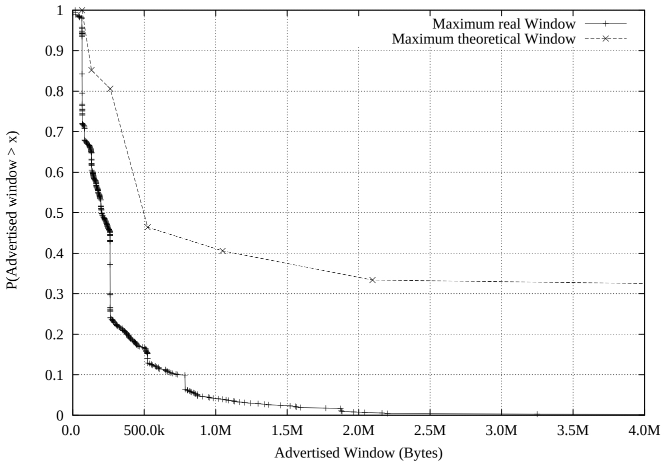

Figure 7. Survival functions of the maximum advertised and allowed windows

However, the calculation of this value is non-trivial in a loaded network as it has to be inferred from packet observed traveling in both directions, taking into account too many parameters such as disorders, retransmission, losses during the capture, etc. RTT can be measured by ICMP probing (Ping) or similar active measurements, but sometimes active injection of traffic is not an option and a passive methodology is preferred.

In this work, a passive methodology is presented to estimate RTT from passive traffic capture. It has been compared to two other classic passive methods: the use of the initial TCP three-way-handshake time, and the observation of the TCP timestamp option defined.

The three estimators are fully passive and can be used on traffic traces.

The proposed algorithm provides an overestimation of the actual RTT value. This is often desired as the values of RTT is used to decide on a timescale above the RTT. The TCP option timestamp estimator has the same property. It never provides a measure lower than the actual RTT but it may give a larger value. On the other hand the initial three-way-handshake RTT may sometimes give smaller RTT samples caused by the presence of middleboxes that answer or establish the connection on behalf of one of the endpoints that is farther away.

It has been shown that in an emulated scenario, the proposed algorithm performs not as accurately as TCP timestamp option method but provides reasonable accuracy. TCP timestamp option method is difficult to improve because it is the observation of an active measurement. The problem with TCP timestamp option method is that the passive estimator requires that the TCP connections are using timestamp option which is not common nowadays.

The proposed algorithm requires that the TCP connection is not filling its $bandwidth \times delay$ product thus, it dependes on the values of RTT and path bandwidth and also on the window scale TCP option but may be used in every connection regardless of its use of timestamp TCP option. Hence it allows the passive estimation of RTT in high bandwidth scenarios (like datacenter networks). In a traffic trace, it provides a RTT estimation for a different set of TCP connections that timestamp option method increasing the number of RTT samples that can be obtained from a captured trace.

Acknowledgment

The authors would want to thank Public University of Navarra for funding through PIF grant.

References

[1] L. Zhang, “Why TCP timers don’t work well,” in Proceedings of the ACM SIGCOMM conference on Communications architectures & protocols, 1986, Stowe, Vermont, United States August 5-7, 1986, 1986, pp. 397–405. [Online]. Available: http://doi.acm.org/10.1145/18172.18216

[2] S. D. Strowes, “Passively measuring tcp round-trip times,” Communications of the ACM, vol. 56, no. 10, 2013, pp. 57–64.

[3] “Tcp extensions for high performance. rfc 1323.” [Online]. Available: https://tools.ietf.org/html/rfc1323 [accessed: 2015-05-29]

[4] B. Veal, K. Li, and D. Lowenthal, “New methods for passive estimation of tcp round-trip times,” in Passive and Active Network Measurement, ser. Lecture Notes in Computer Science, C. Dovrolis, Ed. Springer Berlin Heidelberg, 2005, vol. 3431, pp. 121–134. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-31966-5_10

[5] H. Jiang and C. Dovrolis, “Passive estimation of tcp round-trip times,” SIGCOMM Comput. Commun. Rev., vol. 32, no. 3, Jul. 2002, pp. 75–88. [Online]. Available: http://doi.acm.org/10.1145/571697.571725

[6] G. Lu and X. Li, “On the correspondency between tcp acknowledgment packet and data packet,” in Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement, ser. IMC ’03. New York, NY, USA: ACM, 2003, pp. 259–272. [Online]. Available: http://doi.acm.org/10.1145/948205.948239

[7] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley, “Inferring tcp connection characteristics through passive measurements,” in INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies, vol. 3, March 2004, pp. 1582–1592 vol.3.

[8] “Netem.” [Online]. Available: http://www.linuxfoundation.org/collaborate/workgroups/networking/netem [accessed: 2014-06-01]