Turkish Journal of Electrical Engineering and Computer Sciences

Volume 27 | Number 3 Article 39

1-1-2019

A distributed measurement architecture for inferring TCP round-trip times through passive measurements

Fatih ABUT

Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2019) 27: 2106 – 2120

doi:10.3906/elk-1808-190

A distributed measurement architecture for inferring TCP round-trip times through passive measurements

Fatih ABUT*
Department of Computer Engineering, Faculty of Engineering, Adana Alparslan Türkeş Science and Technology University, Adana, Turkey

Received: 28.08.2018 • **Accepted/Published Online:** 28.01.2019 • **Final Version:** 15.05.2019

Abstract: The round-trip time (RTT), defined as the time elapsed for transmission of a data packet to travel from one endpoint to the other and back again, is an important parameter for Internet quality. This paper proposes an extended version of the well-known SYN/ACK (SA) methodology for passively measuring the RTTs over Transmission Control Protocol (TCP) connections. Differently from the original version of the SA methodology and the rest of studies in the related literature, the proposed passive methodology measures not only the total RTT of an end-to-end connection but also the proportion of the existing connection sections on this entire RTT in a passive way if the connection between client and server is established via intermediate stations. A distributed measurement architecture has been designed that implements the extended SA methodology. Through tests in a controlled laboratory environment, various verification and performance evaluation experiments were conducted to determine the accuracy level of the measurement technique and how the distributed architecture behaves regarding resource requirements as the amount of incoming network traffic increases. Accuracy verification experiments show that on average about 92.66% of the passive measurements are within 10% or 5 ms, whichever is larger, of the RTT that *ping* would actively measure. Furthermore, the results reveal that using today's commodity hardware, the designed distributed architecture exhibits acceptable satisfactory scaling performance and can practically be used to passively measure RTTs of each hop within medium-sized communication networks.

Key words: Round-trip time, passive measurement, transmission control protocol, three-way handshake, quality of service

1. Introduction

In addition to parameters such as throughput and availability for Internet quality, the response time, i.e. the round-trip time (RTT), is a crucial factor. The RTT indicates the time required to send a packet of data from a source to the receiver over a network and to transport the receiver's response back to the transmitter. Measuring and monitoring RTTs in communication networks is important for multiple reasons, such as (a) to investigate and verify service level agreements by measuring the responsiveness of various services, (b) to allow network operators and end users to understand network behavior and performance, (c) to assist in making accurate routing and queuing decisions, and (d) to improve the achievable throughput of transport protocols, to name just a few.

The methods for measuring RTT can mainly be divided into active and passive measurement methods. The active RTT measurement methods are based on injecting additional test packets into the measurement

*Correspondence: fabut@adanabtu.edu.tr

path. A well-known representative of this class is the ping command, which is based on the Internet Control Message Protocol (ICMP). It sends an echo request packet to a destination address that responds with an echo reply packet. The transit time can be estimated from the difference between the arrival time of the echo reply packet and the sending time of the echo request packet [1, 2]. However, there exist several significant disadvantages of active RTT measurements. ICMP is often disabled for security reasons and the additional ICMP packets lead to unwanted additional load of the network, which can falsify the measurements [3]. Active measurement techniques in general suffer from deployment flexibility limitations as all active tools need to be deployed and run on at least one of the end-hosts of each path to be measured, if not on both ends. In the case of only a couple of existing measurement paths, manual deployments and executions of such active tools could lead to temporary solutions without causing a major configuration overhead. However, in the case of reasonably large communication networks that consist of a significantly higher number of paths and hops, their probing via such manual configurations would be practically infeasible. Moreover, active tools report their final estimates in a decentralized way, either on sender or receiver hosts, which makes centralized and uniform network measurement and monitoring a more challenging task. For such a purpose, a distributed measurement architecture is needed that ideally provides the network administrators and operators with a single central host in which the measurements of any path, subpath, or hop within the overall managed network can be flexibly collected without causing any manual configuration overhead.

Passive tools, on the other hand, eliminate the disadvantages of active tools as they do not generate additional traffic, but only capture and analyze the passing real traffic at an appropriate observation point without perturbing the network traffic. They offer the important advantage of flexible placement of the monitoring software anywhere on the way between the end hosts. For these reasons, passive RTT measurement methods are more often preferred in practice [4]. For the passive measurement of RTTs, the well-known Transmission Control Protocol (TCP) is particularly suitable since a transmission of a data segment from the transmitter to the receiver requires the transport of the associated acknowledgment (ACK) segment.

Within the course of the past years, several studies have been carried out to measure RTTs in the literature. Shah et al. [5] investigated the possible relationship between the RTT and the software download time for FTP servers. Günther and Hoene [6] measured RTTs with the intention of determining the distance between WLAN nodes. Jiang and Dovrolis [7] proposed and evaluated a passive measurement methodology that estimates the distribution of RTTs for the TCP connections. The work of Strowes [2] is based on the observation of packets using the TCP header timestamp option, which is used by TCP to generate RTT measures. Aikat et al. [8] measured and analyzed the variability in RTTs within TCP connections using passive measurement techniques. Yan et al. [9] presented a novel passive measurement method that exploits the TCP timestamp option to measure path RTT at an intermediate measurement point between two end-points. Prieto et al. [4] presented a simple passive algorithm to estimate the RTT of a TCP connection in high bandwidth-delay network scenarios. Veal et al. [10] proposed two methods to passively measure and monitor changes in RTTs throughout the lifetime of a TCP connection. Jaiswall et al. [11] proposed a passive measurement methodology to infer and keep track of RTTs with a TCP connection based on the estimated value of the congestion window. Despite the plethora of studies on measuring RTT, the mutual motivation of all these studies was only focusing on measuring or analyzing the total end-to-end RTTs, disregarding the separate calculation of delays over individual hops along that path.

Differently from the rest of the studies in the literature, the aim of this study is to design, implement, and evaluate a distributed measurement architecture that not only measures the total end-to-end RTT. The

delay times of the individual hops should also be measured if the connection between the client and server is established via several intermediate stations, the so-called gateway routers. The idea is to determine the cause of the delay of a packet more precisely. Particularly, the delay of a packet can have different causes, e.g., it can be caused by a congested component within the network of the respective ISP. Another scenario arises if the connection is established via two or more ISPs. In this case, it can be determined which provider along the end-to-end path has delayed the communication. Through tests in a controlled laboratory environment, the accuracy of the proposed passive RTT measurement technique has been validated with the help of active measurements. Also, the performance of the distributed architecture was experimentally evaluated to determine to what extent it is scaling, i.e. how it behaves with respect to capturing arriving frames, connections, and CPU usage as the volume of incoming network traffic is increasing.

2. Theoretical foundations of passive RTT measurement

2.1. End-to-end RTT measurement using SYN/ACK (SA) methodology

There are several well-known methods that can be used to measure the end-to-end RTTs over TCP connections. For example, the slow-start method measures the RTT during the slow-start phase of a TCP connection, while continuous measurement methods measure it over the entire duration of a connection [7]. In this study, the so-called SA methodology is used for passive RTT measurement, which measures the RTT when setting up a TCP connection. Compared to the other two measurement methods, it is easier to implement while also accurately measuring RTTs that are within limits of acceptable accuracy.

Setting up a TCP connection consists of a sequence of three segments, i.e. SYN, SYN/ACK, and ACK segments, and is thus called a three-way handshake (TWH). A monitoring software implementing the SA methodology, placed anywhere on the path between the client and server, measures the RTT by observing the three consecutive TWH segments. Both the arrival times of the initial SYN segment (i.e. active open) and the ACK segment following the SYN/ACK segment (i.e. passive open) are recorded. Subsequently, the end-to-end RTT of the connection can be determined by subtracting the recorded time values. The requirement for a correct measurement is that none of the three segments are lost and the segments are immediately sent by the client or the server [7] (see Figure 1).

The main difficulty in implementing the SA methodology is the filtering of the TWH segments of a TCP connection out of network traffic. The detection of the first two segments of the TWH can be undertaken without problems because the SYN or SYN/ACK flags only set in such types of segments. However, the ACK segment is unidentifiable by its ACK flag as there are segments in the network traffic where the ACK flag is set but they are not part of the TWH phase, e.g., the acknowledgments of data segments, or acknowledgments sent as responses to connection termination requests triggered by one of the end hosts. This problem and its solution will be discussed in Section 3.3 in more detail.

Although the SA methodology can be used to measure the total RTT for the end-to-end connection, its individual usage is not sufficient to calculate the delay times of the individual hops of a connection.

2.2. Proposed method of distributed RTT measurement: extended SA methodology

The placement of the probes on each gateway router allows the computation of the partial RTTs of a connection within a network path. Figure 2 illustrates the principle of distributed measurement through the use of probes on two gateway routers. In Figure 2, the first and second probes are referred to as *alpha* and *beta*, respectively. Using two gateway routers on an end-to-end path results in a total of three sections to be measured, namely

the first hop between the client and the probe *alpha*, the second hop between the probes *alpha* and *beta*, and finally the third hop between the probe *beta* and server.

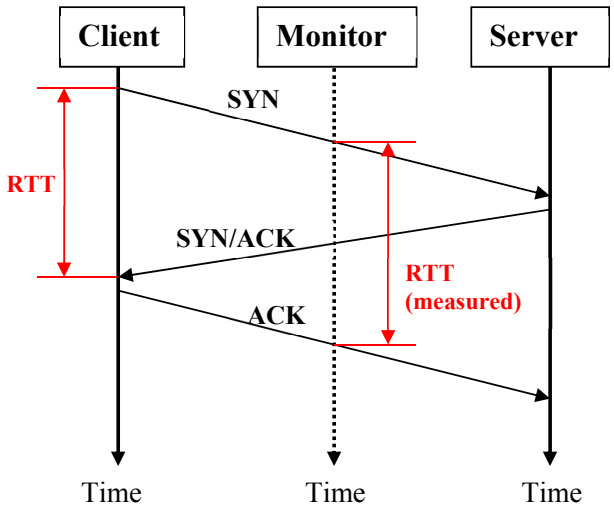


Figure 1. Principle of the SA methodology.

For the RTT calculation of the first hop, probe *alpha* measures the arrival times of both SYN/ACK and ACK segments. Subsequently, the RTT for this hop can be determined by means of a simple difference calculation of these two time values. In Figure 2, this difference value is referred to as Δt_*alpha*-2. In contrast,

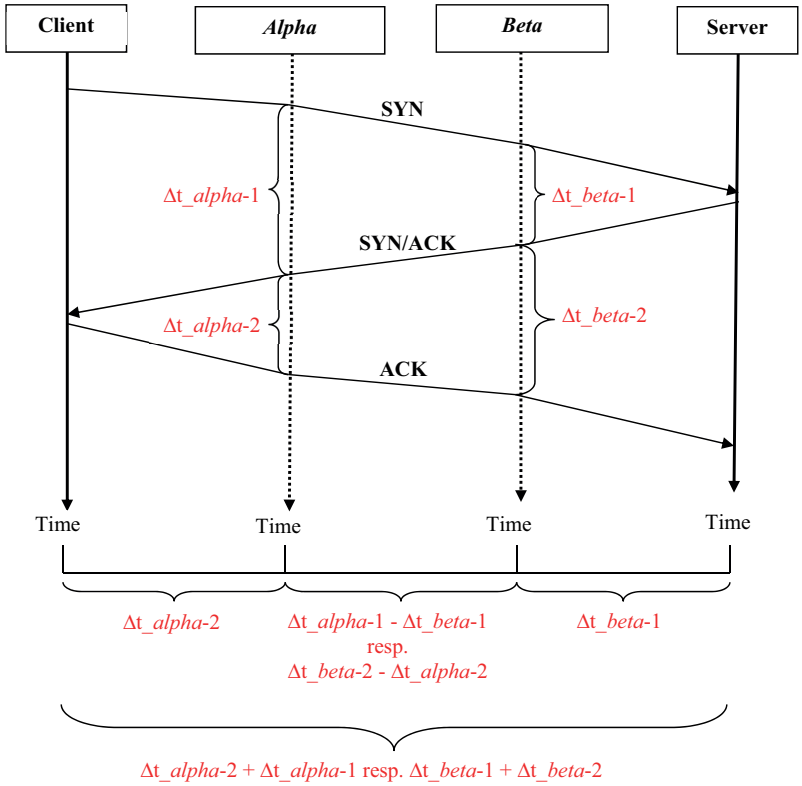


Figure 2. Principle of distributed RTT measurement using extended SA methodology.

RTT calculation of the third hop requires the arrival times of SYN and SYN/ACK segments. Again, the RTT is determined by subtracting the two time values, which is represented by Δt_*beta*-1.

The calculation of the RTT of the second hop can be done in two different ways. On the one hand, the difference t_*alpha*-1 - t_*beta*-1 can be used. In that case, the arrival times of the SYN and SYN/ACK segments are included in the calculation. Alternatively, the difference t_*beta*-2 - t_*alpha*-2 can also be utilized. In that case the arrival times of the SYN/ACK and ACK segments are considered.

The entire route, i.e. the distance between client and server, can, in turn, be calculated by adding Δt_*alpha*-1 and Δt_*alpha*-2 or Δt_*beta*-1 and Δt_*beta*-2.

It is obvious that for the RTT calculation no time synchronization is necessary. To determine RTTs on any hop along the path, the only assumption is the knowledge about when a segment has been sent and at what time the corresponding reply arrives at the transmitter again.

3. Design of the distributed measurement architecture

3.1. Architectural overview

As seen in Figure 3, the concept of distributed measurement architecture, designed to determine the RTTs for each individual hop of an end-to-end TCP connection, is divided into four parts. The first part deals with the acquisition of the TWH segments from the network traffic, which are necessary to calculate the RTTs. For recording these segments, the capture software is responsible, which is also referred to as a probe. For each network component on which this probe runs, a separate database referred to as *ProbeDB* was also created. If probes within the network record such a segment, they send the segment to their respective *ProbeDB*s on the local machines for further processing.

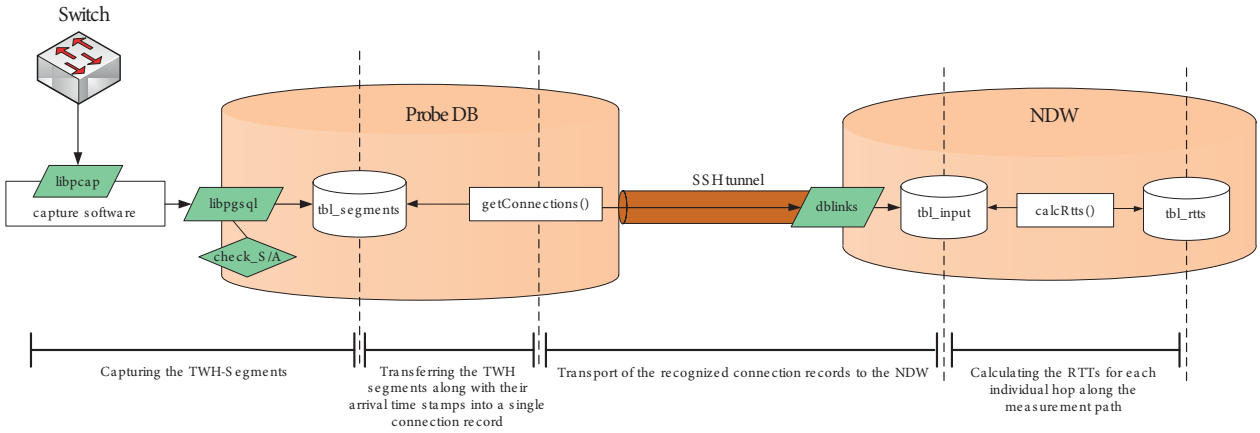


Figure 3. Architectural overview of the distributed measurement architecture.

The structure of the *ProbeDB* is simple. It contains a single table called *tbl_segments*. Recorded data from the capture software are stored in this table. In addition, there is a trigger associated with this table that invokes the *check_count_segments()* function after each insert operation to this table. This trigger function has the task of increasing the counter it manages by one with each call and then checking whether it has reached the selected value 500. This number was determined from the empirical measurements and proved to be a favorable value. If the value 500 is reached, this function calls the *getConnections()* procedure.

The *getConnections()* procedure forms the second part of the designed distributed architecture. Particularly, it transfers the TWH segments stored in the table *tbl_segments* along with their arrival timestamps into

a single connection record. For this purpose, it selects the matching SYN/ACK and ACK segments for each SYN segment found in *tbl_segments*. After successfully mapping and transitioning the three TYH segments into a connection, this procedure generates an "INSERT" statement that transports this connection record to the *sa_input* table of the Network Data Warehouse (NDW). The NDW is the central database in which all the data necessary to calculate the delay times of the hops are collected. The NDW is filled by all available *ProbeDB*s in the monitored network system.

The third part of the designed distributed architecture deals with the transport of the recognized connection records to the NDW. For establishing the connection to NDW, transporting the data, and finally terminating the connection, *ProbeDB* uses the PostgreSQL Database Links extension (dblinks). Database links are modular and provide a variety of functions that allow to execute cross-database SQL statements. The transport of the data between the available *ProbeDB*s and NDW is encrypted via the configured SSH tunnel. Each *ProbeDB* authenticates itself to the NDW using private/public key procedures.

Finally, the *calcRTTs()* procedure forms the fourth and last part, which implements the distributed RTT measurement method presented in Section 2.2. Particularly, to calculate the RTTs of each hop of a measurement path, this procedure first reads the connection records from *tbl_inputs*, applies the extended SA methodology on these data, and finally stores the calculated hop RTTs in *tbl_rtts*.

3.2. Capture software

Before the capture software can be used, an installation and preconfiguration of the database software is necessary. The database software used in this study is PostgreSQL. To connect to PostgreSQL, the probe uses the libpq library. For each recorded frame, it generates an SQL "INSERT" that writes the corresponding header fields to this preconfigured database.

The TWH segments have special flags and contain no payload. To exactly record these segments, the capture software starts with a source-compiled filter expression. This filter expression causes the recording of only those segments having the SYN, SYN/ACK, or ACK flags. However, the distinction of whether or not such a segment contains payload data cannot be uniquely determined with a filter expression. Therefore, this investigation is performed within the capture software itself and can be summarized in three steps; (a) determine the size of the IP packet header, (b) determine the size of the TCP segment header, and finally (c) determine the size of the actual payload by subtracting the total size of the IP packet and TCP segment headers from the IP "Total Length" field. If the difference equals zero, the segment contains no payload data and is therefore assumed to be used for control purposes in TCP connections. In the other case, it represents the size of a data segment and is not transported to the database. Figure 4 illustrates the implementation of this procedure in six steps.

The last ACK segment of the TWH segments cannot be filtered out clearly based on its flag from other ACK segments with payload. Therefore, these unneeded ACK segments are also transported to the preconfigured *ProbeDB*. The recognition and the exclusion of these segments take place in this database.

3.3. Database processing

For connection discovery, the procedure matches the appropriate SYN/ACK and ACK segments for each SYN segment stored in *tbl_segments*. The main question is how individual connection records can be identified within the variety of data stored in the table. Source and destination IPs combined with source and destination ports can be used to identify TCP connections. However, since multiple ACK segments are stored per connection,

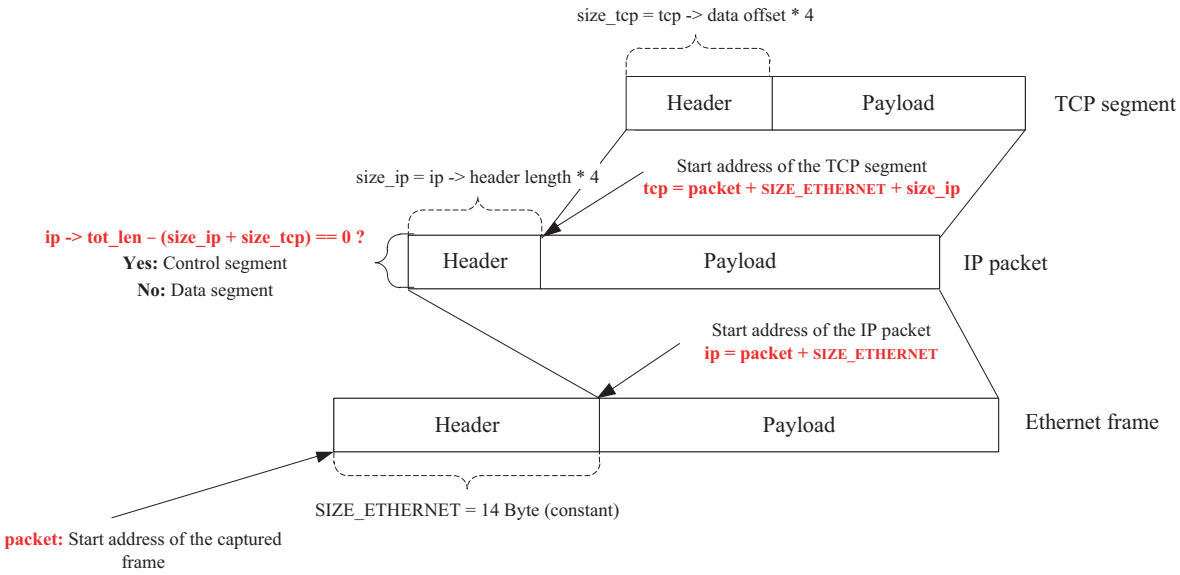


Figure 4. Distinguishing between TCP's control and data segment. The variables *size_ip* and *size_tcp* present the size of the IP and TCP header, respectively. Finally, the variable *ip.tot_len* contains the size of the IP packet, including payload.

the sequence and acknowledgment numbers must also be included to determine the ACK segment of the TWH process. In addition, the same TWH segments for a TCP connection are recorded by each probe in the distributed environment. For this reason, the corresponding identification of the probe that recorded these segments must also be considered when assigning these three segments to a connection. For this purpose, the variable *probe_desc* was introduced, which contains the unique name of a probe.

In order for the corresponding SYN/ACK segment to be assigned to the SYN segment, (a) the source IP address and source port of the SYN/ACK segment must correspond to the destination IP address and destination port of the SYN segment, respectively; (b) the destination IP address and destination port of the SYN/ACK segment must correspond to the source IP address and source port of the SYN segment, respectively; (c) the data record must correspond to a SYN/ACK segment (i.e. fsyn = TRUE and fack = FALSE); (d) the acknowledgment number of the SYN/ACK segment must correspond to the sequence number of the SYN segment + 1; and, finally, (e) the SYN/ACK segment must be recorded by the same probe as the SYN segment.

Conditions (a) and (b) identify the TCP connection between client and server. Conditions (c) and (d) determine the SYN/ACK segment belonging to the SYN segment being processed. Finally, condition (e) ensures that the SYN/ACK segment was recorded by the same probe that also recorded the SYN segment.

Similarly, the corresponding ACK segment can be assigned to the SYN/ACK segment if (a) the source IP address and source port of the ACK segment correspond to the destination IP address and destination port of the SYN/ACK segment, respectively; (b) the destination IP address and destination port of the ACK segment correspond to the source IP address and source port of the SYN/ACK segment, respectively; (c) the data record corresponds to an ACK segment (i.e. fsyn = FALSE and fack = TRUE); (d) the sequence number of the ACK segment corresponds to the acknowledgment number of the SYN/ACK segment; (e) the acknowledgment number of the ACK segment corresponds to the sequence number of the SA segment + 1; and, finally, (f) the ACK segment was recorded by the same probe as the SYN/ACK segment.

Again, conditions (a) and (b) identify the TCP connection. Conditions (c) through (e) assign the appropriate ACK segment to the SYN/ACK segment. Finally, condition (f) ensures that the ACK segment was recorded by the same probe that also recorded the SYN/ACK segment.

The sequence numbers of the SYN and SYN/ACK segments allow the connection records that are transported per probe and per TCP connection to be assigned to each other in the NDW.

After transporting a discovered connection record to the NDW, the TWH segments of that connection are removed from *tbl_segments*. This process repeats until all SYN segments in this table have been processed. After executing the removal procedure, there are three types of segments left over in the dataset: (i) segments of unfinished connections, (ii) unnecessary ACK segments (e.g., data acknowledgments), and (iii) faulty segments or segments of canceled connections. Both the unnecessary ACK segments and the faulty segments or segments of canceled connections can be safely deleted. In contrast, segments of the unfinished connections are still relevant for the RTT measurement. Therefore, it must be ensured that such segments are not deleted. This is achieved by removing only the segments whose timestamps have exceeded at least the timeout of a TCP connection. In the procedure, the timeout of a TCP connection was chosen as 2 min.

4. Testbed and evaluation methodology

4.1. Testbed

As shown in Figure 5, for evaluating the extended SA methodology the traffic between client and web server is monitored. These are interconnected by two switches, resulting in a total of three segments for distributed RTT measurement. The switches each have monitoring capability and are configured to forward or mirror the communication traffic between client and web server to the respective probes *alpha* and *beta*.

The client, router, web server, and NDW, each of which runs the Ubuntu OS, are all equipped with an

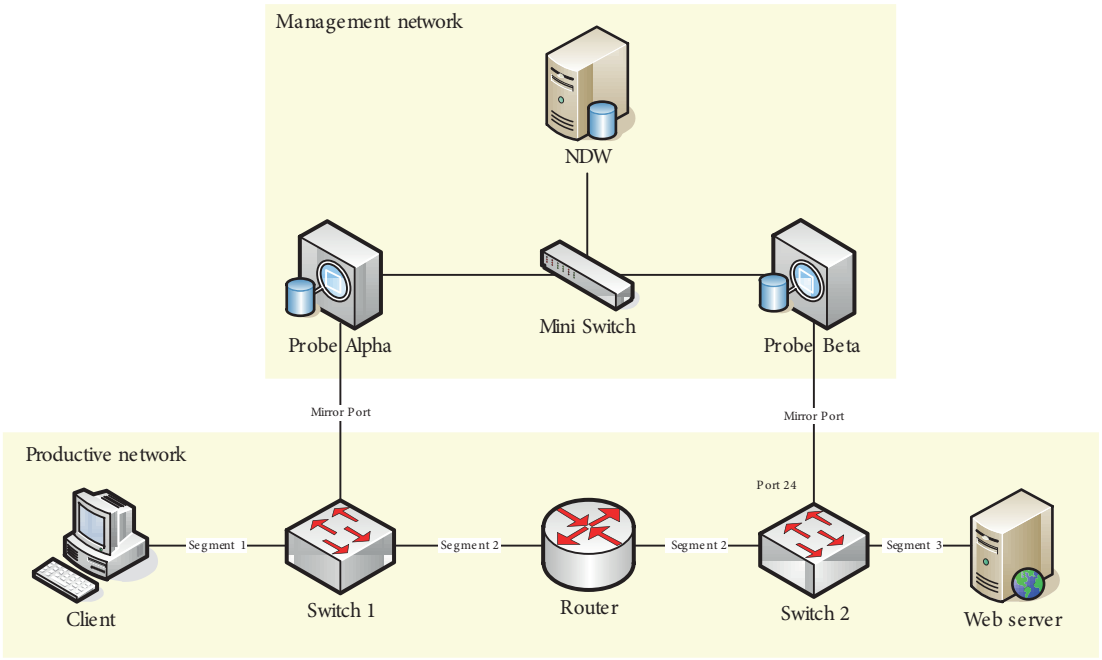


Figure 5. Realization of a distributed environment for passive RTT measurement.

Intel Pentium 4, 2.8 GHz CPU and with 1 GB of RAM. The D-LINK DGS-3224TGR Layer 2 switch providing 24 10/100/1000BASE-T ports for cost-effective copper Gigabit connection was used to connect the participants within the productive and management network. Finally, probes *alpha* and *beta*, also running Ubuntu OS, are equipped with Intel Core 2 1.83 GHz CPU and 2 GB of RAM.

Both on the probes, *alpha* and *beta*, run the capture software that records the TWH segments from this mirrored traffic. The capture software sends each recorded segment to its local *ProbeDB*.

In each created *ProbeDB*, the recorded TWH segments are then transferred with their arrival timestamps into a single connection record and then transported via the mini switch to the NDW.

The concept of the implemented environment envisages the separation of the productive and the management networks. The client and the web server along with switch 1 and switch 2 form the productive data network, while the management network consists of the two probes, the mini switch, and the NDW. The calculation of the RTTs to be measured in the productive network is carried out in this separate management network, so that the transmission and processing capacities of the productive network remain unaffected. Another advantage of this separation is that such an installation of the measurement architecture does not interfere with the original network topology.

4.2. Evaluation methodology

The client and web server in the production network were each configured with a class A address, while the management network used class C addresses. The client uses load generators to generate TCP traffic, which is sent to the web server via the two D-Link switches. In this case, the incoming or outgoing TCP traffic at a particular port is mirrored by each of two D-Link switches via other preconfigured ports to the probes *alpha* or *beta*.

To generate the required TCP traffic, two different load generators were used. The first load generator is called the Distributed Internet Traffic Generator (D-ITG) [12] and allows the generation of individual TCP segments. D-ITG is controlled by command lines and can generate segments with a maximum data rate of 600 Mb/s, which is sufficient for carrying out the measurements. As the second load generator, the Apache Benchmark tool (ABT) coming with the Apache web server was chosen. Unlike D-ITG, ABT enables the establishment of TCP connections with any number of preconfigurable repetitions. The parameterization of this benchmark software is also done via the command line.

The measurements to be performed were divided into two parts. In the first part, the accuracy of the proposed distributed RTT measurement technique is to be evaluated. To this end, a similar experimental methodology used in [7] was applied in this study. Particularly, as the SA methodology examines the TWH phase of a TCP connection, the active *ping* measurements, the results of which are used for reference purposes, are collected just before the connection establishment.

The Apache web server illustrated in Figure 5 forms the measurement target. An available HTTP file, larger than 10 KB, is provided by the web server. First, to actively measure the RTTs of the three hops (i.e. source, transit, and destination subnetworks) individually, which will represent the reference values for the SA estimations, a shell script was written that automatizes logging in to the client and routing machines via SSH using the public key principle to trigger the active hop measurements in parallel. Immediately after the RTTs for the three hops have actively been measured, the shell script causes the client to transfer the HTTP file from the target using the GNU Wget utility over the same path. After the transfer is over, the RTTs of the HTTP-TCP transfer for the three hops are passively estimated by the measurement architecture using the distributed SA

estimation method described in Section 2.2. Finally, the RTTs for the three hops passively estimated by the proposed technique are compared to the average of the ten active *ping* measurements.

The experiments have been repeated with 200 different randomly set queuing times. Particularly, in the client, router, and web server machines, the delay of packet forwarding has been controlled using the Netem tool, which was configured to cause queuing delay variations between 10 ms and 500 ms to simulate real network conditions. For each of the randomly set queuing times, the average of the 10 *ping* measurements was used as a basis for comparison with the SA estimates.

The second part of the measurements concerned aspects related to how the system scales in terms of arriving frames, TCP connections, and processor usage. The scaling experiments, in turn, were performed in two groups. The first group of the scaling experiments aims to determine how many individual TCP segments per second the capture software can record maximally. To carry out this measurement, the load generator D-ITG was used. During these measurements, the processor utilization generated by the capture software and postgres processes was also observed. The Unix tool top was used to measure the processor load. The measurements in the first part were carried out as follows: the D-ITG load generator was parameterized to send a certain number of TCP segments per second to the D-ITG receiver. After the experiment, *tbl_segments* in the database was checked to see how many of these sent segments could be saved by the capture software.

In contrast to the first group, the second group of experiments was related to the generation of TCP connections instead of TCP segments, to make a benchmark of to what extent the proposed measurement architecture is scaling. More specifically, it should be determined how many of these generated TCP connections could completely be captured by the capture software to infer the RTTs from the TWH segments for each hop. To this end, the well-known ABT from the Apache Foundation was used to perform these measurements. It generates load in the form of HTTP requests to a web server. The program also supports simulating concurrent access to a web server. The measurements were carried out similarly as in the first part. ABT was instructed to establish a certain number of TCP connections to the web server. The number of detected connections was then determined using the *getConnections()* procedure in the *ProbeDB*, and finally compared with the number of connections generated by the ABT.

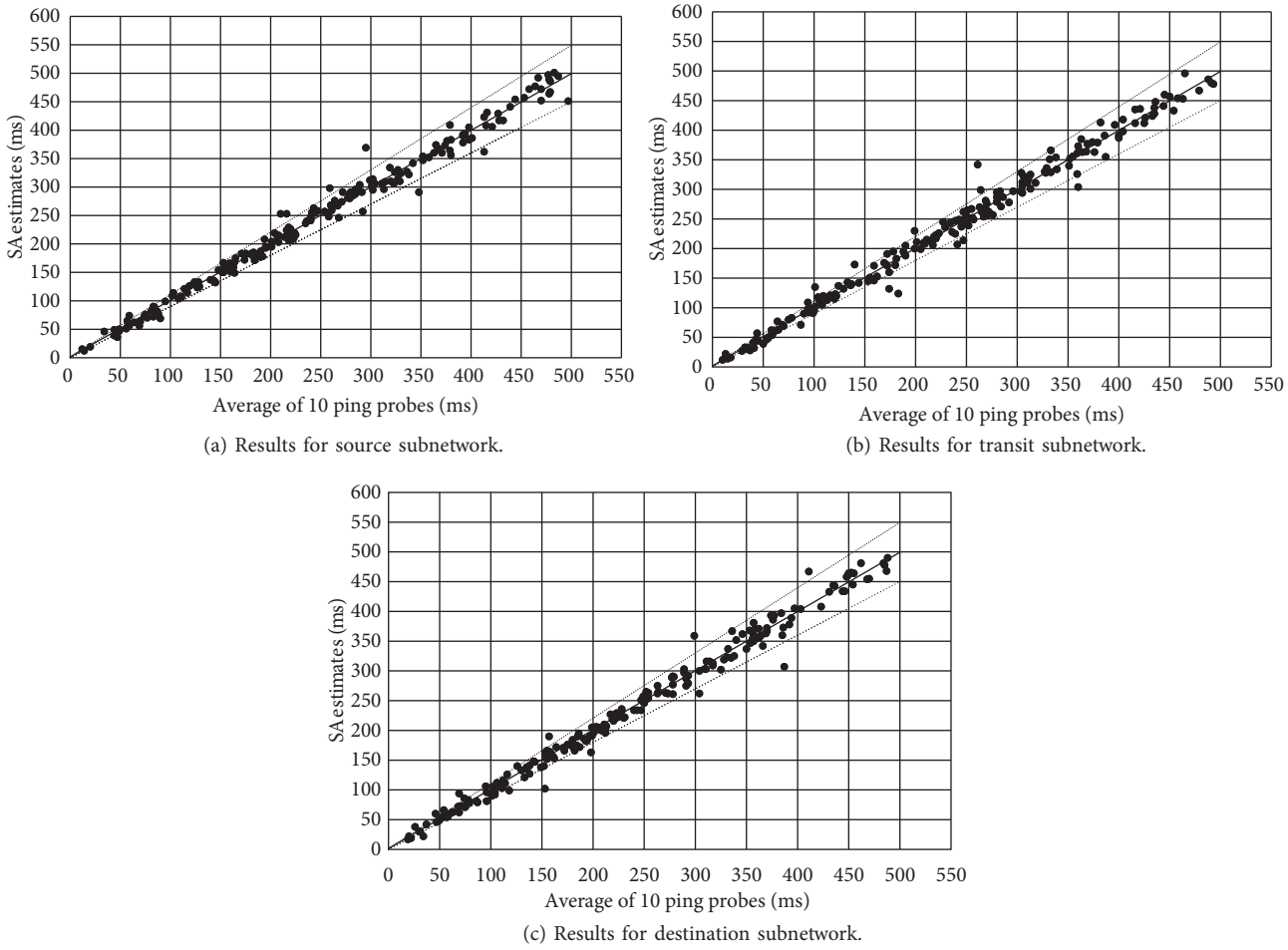
5. Results and discussion

This section includes two subsections. In the first one, the accuracy of the extended SA methodology is verified by directly comparing the passive RTT measurements with active ones using the *ping* command. In the second one, the results of scaling experiments for the implemented measurement architecture are presented.

5.1. Verification of measurement accuracy

Figures 6a–6c illustrate the distributed SA estimates in comparison with the corresponding *ping* measurements for the source, transit, and destination subnetworks, respectively. Analogously to the methodology used in [7], an SA estimate for a hop is categorized as accurate if it is within 5 ms or 10% of the corresponding average active *ping* measurement. The dashed lines in Figure 6 illustrate this accuracy region. With this error tolerance it is observed that the fraction of inaccurate measurements of the proposed extended SA methodology is 6%, 9%, and 7% for the source, transit, and destination subnetworks, respectively.

By using the well-known Wilcoxon signed-rank test [13], it has also been investigated whether the difference between active and passive estimates is statistically significant or not. More specifically, the test has been applied on three different pair sets including the (*ping*, SA)$_{source}$, (*ping*, SA)$_{transit}$, and (*ping*,



(a) Results for source subnetwork.

(b) Results for transit subnetwork.

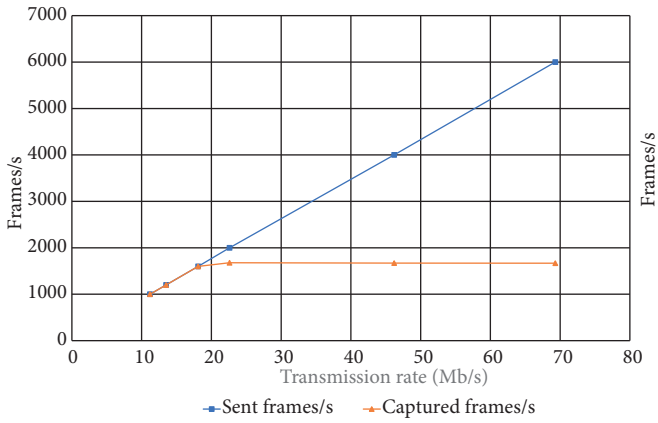
(c) Results for destination subnetwork.

Figure 6. Comparison of the distributed SA estimates with the average of the *ping* measurements just before the TCP connection establishment for the source, transit, and destination subnetworks.

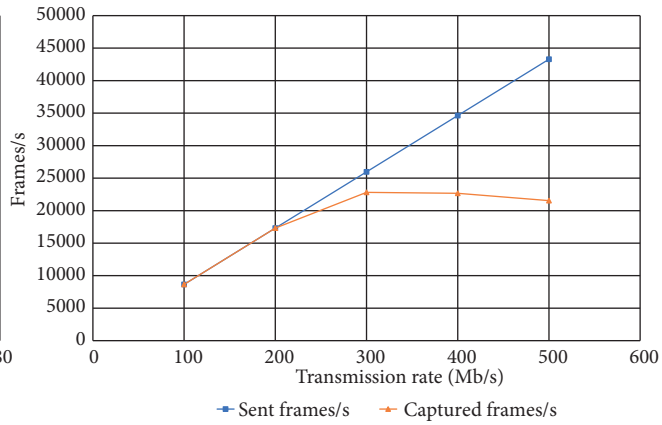
SA)$_{dest}$ pairs, which represent the corresponding active and passive estimates of RTTs for the source, transit, and destination subnetworks, respectively. The sample size in each test case equals 200, and the two-sided level of significance, i.e. α, is set to 0.05. The W values for (*ping*, SA)$_{source}$, (*ping*, SA)$_{transit}$, and (*ping*, SA)$_{dest}$ are 9442.0, 9030.5, and 8034.5, respectively. Since the sample size is greater than 20, the table of critical values for W cannot be utilized [14]. Alternatively, the statistical analysis can be conducted using the normal distribution approximation. In this case, the required calculations for (*ping*, SA)$_{source}$, (*ping*, SA)$_{transit}$, and (*ping*, SA)$_{dest}$ yield z scores of –0.0198, –0.6647, and –1.1357, respectively, which in turn yield P-values of 0.98, 0.51, and 0.25 for $\alpha = 0.05$. All P-values are bigger than α; therefore, the null hypothesis is accepted. In conclusion, the test shows that there is a statistically insignificant difference between actively and passively estimated values of RTTs for all three subnetworks of the measurement path.

5.2. Scaling experiments

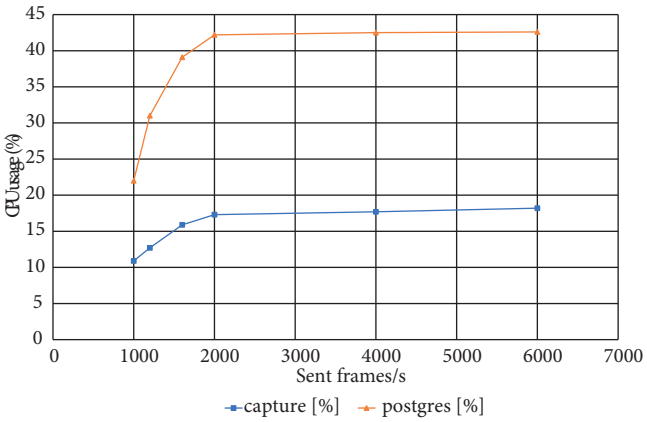
Figure 7a illustrates the results for the first part of the scaling measurements, i.e. how many frames per second the capture software can write to the database. It is to be noted that the capture software does not write the entire frame content into the database, but only the special fields from IP and TCP headers that are needed for



(a) Writing the captured frames into the database.



(b) Writing the captured frames in a local file.



(c) CPU usages of capture software and postgres processes.

Figure 7. Results of scaling experiments.

RTT calculation. The measurements took place over a period of 1 s. A comparison measurement over a longer period showed no change in these results.

The results show that the capture software can, to a certain arrival rate, capture the frames sent by D-ITG with no losses. However, at that particular frame arrival rate (i.e. starting from approx. 22 Mbit/s), the number of recorded frames stops increasing and remains at a stable level. This shows that with a payload of 1448 bytes, the capture software can record a maximum of approximately 1600 frames per second. At this point, the question arises of why not all transmitted frames can be captured. The first cause for this performance degradation has been revealed to be the libpcap library, which was utilized in the capture software. More specifically, the libpcap library at one time can either record the incoming frames or can send a recorded frame into the database. More specifically, while writing a recorded frame to the database, it cannot capture the newly arriving frames during this writing process.

The second cause of frame losses is that each captured frame is directly written to the database by the capture software. Inserting each recorded frame into the database creates a heavier system load and can also cause frame losses. To verify this, the capture software was manipulated to write each recorded frame to a local file instead of the database. The measurements, illustrated in Figure 7b, show that up to approximately 22,000 frames can be captured per second, while that number when writing to the database is only approximately 1600

frames per second, as shown in Figure 7a. Thus, it can be concluded that writing the recorded frames to the database is the second major cause of frame losses.

During the measurements it was observed how the processor load of the two processes, namely the capture and postgres processes, was related to the data rate. Figure 7c shows the average processor utilization of these two processes. It is seen that the percentage processor utilization of the two processes increases in total up to approximately 60%, but thereafter this value is no longer exceeded. Most of the processor utilization is generated by the postgres process. It alone causes processor load up to about 42%, while this value in the capture process is about 18% in maximum.

In contrast to the first part, the second part of the scaling experiments concerned the generation of TCP connections instead of TCP segments, investigating how many of these established connections could be captured by the capture software for a successful calculation of per-link RTTs. First, a single HTTP GET request was sent to the web server, which consequently established a single TCP connection. Payload data of 1448 bytes (exactly one data segment) were transported per connection. Afterwards, the recorded and supplied data of the probes in the *ProbeDB* were checked for plausibility. Through this single TCP connection, the capture software recorded a total of six segments. The first three of these six segments were the TWH segments of the TCP connection setup required for RTT measurement. The remaining three segments are acknowledgments, which are indistinguishable from the recorded TWH segments by their ACK flag (i.e. data or connection termination acknowledgments). More specifically, for each individual TCP connection consisting of a connection setup, phase, and termination, the capture software collects a total of six segments.

The measurement results illustrated in Figure 8 show that the capture software was able to capture up to 100 back-to-back established TCP connections without any losses. After that amount, the first frame losses appear. The same tests were also repeated with a concurrency of 5, 10, and 15 clients, i.e. several back-to-back connections from 5, 10, and 15 clients were established simultaneously. In this case, the capture software could process up to approx. 75, 50, and 25 connections concurrently established by 5, 10, and 15 clients, respectively, without any losses. However, it is clearly seen that the simultaneous establishments of connections to the web server has significantly decreased the number of captured connections compared to the case of establishing the connections sequentially. If, e.g., via the capture software 100 TCP connections are established sequentially, all of them could be captured by the capture software. If, on the other hand, the 100 TCP connections are started with a concurrency of 15 clients, the capture software could only record 52 connections.

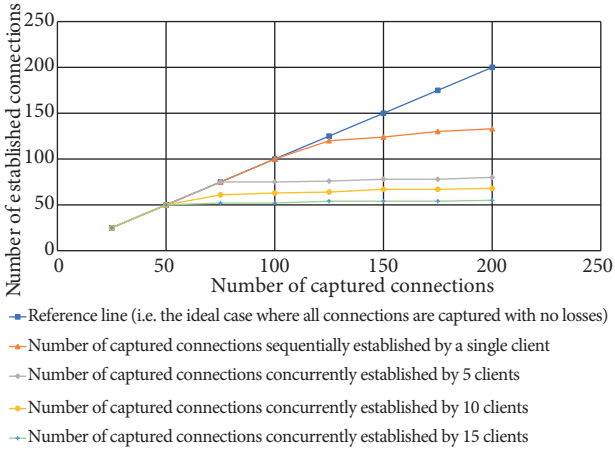


Figure 8. Number of successfully captured connections sequentially established by a single client as well as with a concurrency of 5 and 10 clients.

The scaling experiments have been carried out with the primary intention of determining to which extent the implemented distributed measurement architecture is scaling with respect to processing the arriving connection requests. The difficulty in making this decision is determining when the system begins to scale. Considering a network where the capture software maximally receives 100 sequentially (i.e. back-to-back) established connections, or 50 connections simultaneously established by 10 clients, it can be concluded that the system will scale. On the other hand, if, e.g., over 100 sequential or 50 simultaneous connections are established, the system will not fully scale. The question here is therefore to find a general statement about typical TCP traffic in the network that is to be monitored using the distributed measurement architecture.

6. Conclusion and future work

This paper proposed an extended version of the well-known SA methodology for passively measuring RTTs over TCP connections. In contrast to the original version of the SA methodology, which can only measure the end-to-end RTT of a TCP connection, the proposed extended version of the SA methodology can determine the RTTs for each individual hop of an end-to-end TCP connection. A distributed measurement architecture has been designed that implements the extended SA methodology. Through tests on a real testbed with different levels of loads and queuing delays, various verification and performance evaluation experiments were conducted to determine the accuracy level of the measurement methodology and how the distributed architecture behaves regarding resource requirements as the amount of incoming network traffic increases. The results show that in terms of accuracy, about 92.66% of the passive measurements are within 10% or 5 ms, whichever is larger, of the RTT value that ping would actively measure. Also, the experimental results reveal that using today's commodity hardware, the measurement architecture exhibits acceptable satisfactory scaling performance and can practically be used to passively measure RTTs of each hop within medium-sized IP-based networks.

However, during experimental evaluations, two issues were identified that inhibit the architecture from processing a larger amount of TCP connections, and these should be addressed in future work. The first issue is caused by the usage of the libpcap library, which can either record the arriving frames or write a recorded frame to the database at a time. One possible solution to this problem is to decouple the two operations of monitoring the incoming frames and writing them to the database. The second issue is that writing a recorded frame to the database causes about twice the processor utilization compared to the actual capture software due to the constant transport process of the captured data. Particularly, the capture software generates its own SQL "INSERT" command for each recorded frame and passes it to the database. Instead, one could first locally inject the SQL "INSERT" commands, e.g., in a RAM disk, and then transfer all cached data to the database at once.

Acknowledgments

The author would like to acknowledge the helpful advice and support of Prof Dr Martin Leischner, to whom the author is greatly indebted. The author would also like to thank A Crngarov, A Neth, C Niephaus, K Reineck, and P Weber, who were partly involved in the initial development of the capture software and *ProbeDB* design.

References

[1] Mnisi NV, Oyedapo OJ, Kurien A. Active throughput estimation using RTT of differing ICMP packet sizes. In: IEEE International Conference on Broadband Communications, Informatics and Biomedical Applications; 23–26 November 2008; Gauteng, South Africa. pp. 480-485.

[2] Strowes SD. Passively measuring TCP round-trip times. Communications of the ACM 2013; 56 (10): 57-64. doi:10.1145/2507771.2507781

[3] Zander S, Armitage G. Minimally-intrusive frequent round trip time measurements using synthetic packet-pairs. In: IEEE Conference on Local Computer Networks; 21–24 October 2013; Sydney, Australia. pp. 264-267.

[4] Prieto I, Izal M, Magaña E, Morato D. A simple passive method to estimate RTT in high bandwidth-delay networks. In: Seventh International Conference on Evolving Internet; 12–18 October 2015; St. Julians, Malta. pp. 6-11.

[5] Shah SMA. Measuring round trip time and file download time of FTP servers. Network and Complex Systems 2011; 2 (5): 8-14.

[6] Günther A, Hoene C. Measuring round trip times to determine the distance between WLAN nodes. In: Proceedings of Networking; 2–6 May 2005; Waterloo, Canada. pp. 768-779.

[7] Jiang H, Dovrolis C. Passive estimation of TCP round-trip times. ACM SIGCOMM Computer Communication Review 2002; 32 (3): 75-88. doi:10.1145/571697.571725

[8] Aikat J, Kaur J, Smith FD, Jeffay K. Variability in TCP round-trip times. In: ACM SIGCOMM Conference on Internet Measurement; 27–29 October 2003; New York, NY, USA. pp. 279-284.

[9] Yan H, Li K, Watterson S, Lowenthal D. Improving passive estimation of TCP round-trip times using TCP timestamps. In: IEEE International Workshop on IP Operations and Management; 11–13 October 2004; Beijing, China. pp. 181-185.

[10] Veal B, Li K, Lowenthal D. New methods for passive estimation of TCP round-trip times. In: 6th International Conference on Passive and Active Network Measurement; 31 March–1 April 2005; Boston, MA, USA. pp. 121-134.

[11] Jaiswal S, Iannaccone G, Diot C, Kurose J, Towsley D. Inferring TCP connection characteristics through passive measurements. In: IEEE INFOCOM; 7–11 March 2004; Hong Kong, China. pp. 1582-1592.

[12] Avallone S, Emma D, Pescapé A, Ventre G. Performance evaluation of an open distributed platform for realistic traffic generation. Performance Evaluation 2005; 60 (1-4): 359-392. doi:10.1016/j.peva.2004.10.012

[13] Taheri SM, Hesamian G. A generalization of the Wilcoxon signed-rank test and its applications. Statistical Papers 2012; 54 (2): 457-470. doi:10.1007/s00362-012-0443-4

[14] Anaene Oyeka IC, Ebuh GU. Modified Wilcoxon signed-rank test. Open Journal of Statistics 2012; 2 (2): 172-176. doi:10.4236/ojs.2012.22019