TUHH
Technische Universität Hamburg-Harburg

ComNets
Institute of Communication Networks

Master's Thesis

Evaluation and implementation of a network performance measurement tool for analyzing network performance under heavy workloads

Kuralamudhan Ramakrishnan

Hamburg, 04 May 2015

Evaluation and implementation of a network performance measurement tool for analyzing network performance under heavy workloads

Kuralamudhan Ramakrishnan
Immatriculation number: 21263139
Program: Information and Communication Systems

Hamburg University of Technology
Institute of Communication Networks

First examiner: Prof. Dr.-Ing. Timm-Giel
Second examiner: Dr.-Ing. habil. Rainer Grünheid
Supervisor: Dr. Alexander zimmermann, NetApp

Hamburg, 04 May 2015

Declaration of Originality

I hereby declare that the work in this thesis was composed and originated by myself and has not been submitted for another degree or diploma at any university or other institute of tertiary education.

I certify that all information sources and literature used are indicated in the text and a list of references is given in the bibliography.

Hamburg, 04 May 2015

Kuralamudhan Ramakrishnan

Acknowledgment

This thesis is carried out as part of M.Sc degree program in Information and Communication System in Hamburg University of Technology (TUHH). The thesis work is done with the collaboration of Institute of Communication Networking at TUHH and NetApp GmbH Research Lab in Munich

I would like to take this opportunity, to thank Prof. Dr.-Ing. Andreas Timm-Giel, Vice President for Research and Head of Institute of Communication Networks at TUHH, for his support during the course of this project and my NetApp supervisor Dr. Alexander Zimmermann for his vital support and guidance through the study. He provided me the platform to work in the advanced technologies and also guided me with valuable advice and technical support. I also like to extend my special thanks to the Advanced Technology Group members Dr.Lars Eggert, Dr. Doug Santry and Dr. Michio Honda for guiding me throughout the project work.

Abstract

Transmission Control Protocol (TCP) is the most prominently used transport layer protocol in the Internet[26]. The Internet performance experienced by the users of all Internet applications totally relies on the efficiency of the TCP layer. Hence, understanding the characteristics of the TCP is vital to properly design, employ, and evaluate the performance of the Internet, for the research work based on network design. So the aim of the project is to study the TCP performance to provides information on the speed and reliability of an unreliable network present in modern Internet. On the quest of this issue, today's research is focused on the Internet community equipped with a number of performance metrics measurement tool.

With the emerging Internet growth, the ultra-high speed data connections are no longer considered as the greater achievements for the modern communication networks. Gigabit Ethernets are already being dominated with 40 and 100 Gbit/s data rates [61], which are standardized in Institute of Electrical and Electronic Engineers (IEEE) 802.3bj [7]. The recent development in the latency sensitive application in the communication networking arise the concerns about latency measurement. Unfortunately there are little or no research tools available to calculate precisely the latency in the network, according to the standard procedure.

This thesis explores the latency measurement in standard methodology using the Internet Engineering Task Force (IETF) standard procedure [6] and aims to implement the latency performance measurement tool using the Linux kernel time stamping[43] feature. Subsequently, their performances are evaluated. The result shows that latency measurement at different stacks are affected by the network stack, device drivers and by the user space, than the actual latency in the network.

Contents

1 Introduction

1.1 Motivation

The TCP was born to avoid the congestion in the network and ensure goodput, quality of service, and fairness. The traditional measure of network is expressed in the terms of bandwidth. In Open System Interconnection (OSI) – 7 layer model, the end users of the traffic model in a network are the application layer. Each request submitted to the computer must be done within a particular time frame, whereas the applications like file download, email exchange are not sensitive to per packet delivery time. These are categorized as the throughput-oriented application. But in recent applications like VoIP, interactive video conferencing, network gaming, and automatic algorithmic trading, the per packet delivery time is very important. These applications involve humans and machines, where operations are multiple parallel requests and response loops involve thousands of servers. Currently we are in a scenario where the measurement of precise latency of individual request ranges from milliseconds to microseconds. So the data centers are now focusing in areas toward the improvement of low latency in their network infrastructure [3].

Traditionally for a decade, the primary focus in terms of performance metrics in the networking has been bandwidth. With growing capacity in the Internet, bandwidth is not the main concern anymore [11]. The network – induced the latency as one-delay [4] or Round Trip Time (RTT) [5] often has noticeable impact on the network performance [11].

We are inclined towards the accuracy and reliability of the latency measurement as a primary metric for evaluating the current generation networks and data center. A significant amount of storage, computing and communication are shifting towards data centers [3]. Within a boundary scope, data centers are characterized by propagation delays, delay in network stack, queue delay in the switches. Delivering predictable low latency appears more tractable than solving the problem in the internet at large.

Applications like high performance computing, and RAMCloud [44] [46] involves multiple parallel requests – response loop and remote procedure calls to thousands of servers. Platforms like RAMcloud integrates into the search engine and social networking, and such an application must share the network with throughput-oriented traffic which consistently deals with terabytes of data [3]. So measuring the latency in this heavy loaded condition is absolutely necessary.

Traditionally latency is measured with Internet Control Message Protocol (ICMP) ping. Although ICMP is a great way to check for the link availability, it is not the standard way to test for latency or delay in a network. The ping uses a series of ICMP, echo messages to determine the link availability, round-trip delay in communication with the destination devices. The ICMP ping command sends an echo request packet to destination address, then waits for a reply and the destination sends an echo reply back to the source within a predetermined time called a *"timeout"*. The default time out duration depends on the routers [32].

ICMP message are considered to be low priority messages, so the routing platform will respond to other higher priority messages, such as routing updates. The kernel also introduces tens of milliseconds of processing delay to ICMP message handling and even these delays are not uniform. ICMP ping latency is not a recommended way of testing latency in the network. One of the accurate and best ways to calculate the latency is to stimulate data traffic, using the traffic generator for the transit traffic [32].

1.2 Objective

The main objective of the thesis is to standardize and define a particular metrics (latency) to be developed under the general framework developed by the IETF [6], IP Performance Metrics (IPPM) [53] of the Transport Area.

The thesis begins by laying out criteria for the latency metrics that has been adopted from the IPPM. These criteria are designed based on the IPPM standards and methods that will maximize an accurate common understanding regarding metrics definition.

This project then defines the fundamental concepts of latency metrics and measurement methodology, which clearly explains about the measurement issues. Given these concepts, the latency measurement uncertainties and errors are discussed later.

The latency metric is defined in the two metrics one – way delay [4] and round-trip delay [5] of packets across the internet paths. There are separate RFCs for each metric. In some cases, IPPM working group mentions that there might be no obvious means to effectively measure the metrics. Hence difficulty in practical measurement is sometimes allowed, but ambiguity in meaning is not allowed [60].

The measurement methodology for the latency metrics are defined in the IPPM RFC, but for a given well defined metrics there might be different measurement methodology as defined in the RFC 2330 [60]. The methodology for a metric should have attributes and principle that the methodology is repeated multiple times in the similar condition and their results are consistent [60].

Even the best methodology for the measurement metrics would result in errors. So the measurement tool should understand the source of uncertainties/errors, and also should minimize and quantify the amounts of uncertainties / errors [60]. The derived

metrics [2] and metrics by spatial [25] and temporal composition [2] cause measurement uncertainties.

Measurement of time plays a crucial part in designing a methodology for measuring a metric, where the uncertainties/errors are introduced by the imperfect clock synchronization. So the objective of this project is to standardize the performance metrics according to the standards and property mentioned in the IPPM RFC's.

Implementation of latency measurement under load measurement, unlike ICMP ping,is done by looking in the adaptation of open source TCP/IP measurement tools, which provides a platform for the effective and efficient way to stimulate the TCP traffic under the heavy load condition. Evaluating the fairness and correctness of the measurement is carried out using different network load and using different stimulation scenarios to justify the correctness, and accuracy of latency measurement. Then the evaluation the performance metrics by comparing the results of existing measurement tools is done.

1.3 Approach

In this section, the general approach for the latency measurement under the heavy load condition will be discussed. As mentioned in the section 1.2, the latency measurement is incorporated as a separate module in the open source existing measurement tools. The existing standard open source measurement tools for the TCP/ IP measurements are Iperf [33], Netperf [41], Thrulay [58], TTCP [13], NUTTCP [42].

These measurement tools are compared based on their features, architecture, feasibility in adopting the latency measurement, performance, supported protocols, Central Processing Unit (CPU) utilization, directionality , Inter process communication, metrics supported, user interface(Command line Arguments), fairness issues, socket options, and QoS.

The details regarding all these features and also comparison between the measurement tools will be discussed in the chapter 3. Based on the comparison of the features between these tools, the new measurement tool flowgrind is found to be superior to the other standard measurement tools in terms of architecture and the feasibility of implementation of latency measurement, under heavy loaded condition.

The methodology of the latency measurement is based on the standard definition of one-delay [4] and two-delay [5] as mentioned in the IPPM standard working group of IETF.

The implementation of the latency measurement is done by using the linux timestamping option available in the Linux kernel. Linux timestamping is used to timestamp each and every event handling of data buffer in the Linux kernel and in addition to it, it also handles the timestamping in the Network Interface Controller (NIC). These kernel level timestamping are mentioned as software timestamping and the Network interface

level timestamping are mentioned as the hardware timestamping. These hardware timestamping are used to record the received timestamping and transmit timestamping via network adaptor. Linux kernel timestamping or software timestamping supports more event timestamping than the network adaptor hardware timestamping. The software timestamping support both to receive and transmit timestamping, it also supports data buffer acknowledgment, packet scheduler timestamping, which will be discussed in detail regarding these timestamping in the section 6.1.

1.4 Overview

The thesis is organized as follows: In chapter 2, the basis standard procedure to measure the performance metrics as mentioned in the IETF, the sections discuss in details regarding the working groups, and standard procedure to measure the one-delay and two way delay. In chapter 3, discuss regarding the current performance measurement tools and discuss in details regarding their merit and demerit. The chapter 4 gives the general idea regarding the traffic generation and discuss in details regarding the stochastic traffic generation and the traffic model. The chapter 5 introduces the flowgrind, and explains the merits and advantage of developing the latency measurement module in the flowgrind. The chapter 6, explains the actual implementation of the latency measurement module in the flowgrind using the Linux timestamping feature. In chapter 7 evaluate the latency measurement using stochastic traffic generation. Finally chapter 8 give insight regarding the potential future work.

2 Standardization of performance

2.1 IETF

The standardization of protocol and procedure is carried out by the Internet Engineering Task Force (IETF), which is considered as international platform for the network engineers, operator, designer and research community. The main activity of the IETF is to standardize the internet infrastructure and operation procedure [6]. Any person can contribute their work to the IETF. The technical competence work is done in its working group, through the working group mailing list [6]. Let us look into the *"standard"* definition in the IETF from the Request for Comment (RFC) 3935. Standard is the term, which define a specification of a protocol, procedure or system behavior, *"if you want to do a certain thing, and this gives the description of how to do it"* [6]. But it is not mentioned to use procedure as the mandate or the compulsory one. It is only mentioned in the RFC 3935 that if someone says that he or she is doing his/her research work according to this standard, it benefits interoperability in the internet. The multiple products, which are implemented according to the standard procedure, can work together and which could be used widely as valuable functions to the Internet users [6].

As mentioned earlier in the section 1.4, the IETF has many working group for several research area. In this chapter, the two working group for the performance metrics standardization and the bench marking methodology will be discussed.

2.2 BMWG

The Benchmarking Methodology Working Group (BMWG) from the IETF charter [53], recommends the standardization of the metrics mainly for the internetworking technologies. Internetworking technology devices are mainly network router, switch, services and system. The recommendation of standardization for a class of equipment involves the discussion of performance metrics that are apt to that class. The BMWG differentiate itself from other working group for the performance metric measurement by limiting its scope for the internetworking technology [53]. It means that their performance metrics methodologies are not applicable to the benchmarking functional networks. It is applicable only to the controlled laboratory networks. BMWG works closely with the network operators, network test tool developers to do the benchmarks that are independent of the vendor specific and testing methodology is applicable universally to the all internetworking technology class [53].

Data center benchmarking in the BMWG is used to evaluate the data center performance. This benchmarking includes the network congestion scenarios, data center switch buffer analysis and traffic conditions. The RFC 1242 [8], defines the latency definition in the terms of store and forward devices , *"the time interval starting when the last bit of the input frame reaches the input port and ending, when the first bit of the output frame is seen on the output port"* [8]. And for the bit forwarding devices, *"The time interval starting when the end of the first bit of the input frame reaches the input port and ending when the start of the first bit of the output frame is seen on the output port"* [8]. The RFC 2544 [48] gives the methodology to implements the definition as mentioned in the RFC 1242.

The RFC 2544 discusses the throughput measurement, latency and frame loss rate for the networking devices and used for benchmarking the performance metrics for the data plane for the networking interconnection device. But the RFC 2544 defines the pre-defined frame size for testing. This procedure is tested by using the IxCloudPerf QuickTest [34], the main objective of this testing is to test the client – server and server - server traffic. The switches are tested with both the client – server traffic and server – server traffic with different frame size packet. The results are different for different frame size, traffic patterns and also affect the latency and throughput. So this is considered as the disadvantage and shows the inefficiency of the RFC 2544 testing methodology.

2.3 IPPM

From the IP Performance Metrics (IPPM) working group from the IETF charter [53], IPPM is working in developing and maintaining the standardizing of the performance metrics, that could be applied to the performance and reliability of the application that running over the transport layer protocols (For Example, Transmission Control Protocol (TCP), User Datagram Protocol (UDP)) over IP and it is out of scope for IPPM to work on the metrics that are applicable to the lower layer Ethernet Operations, Administration and Management(OAM) mechanism. This is the main difference between the BMWG and IPPM, and the methodology of measuring metrics in both the working group reflects this objective. For example, in BMWG it deals with the Ethernet frame for the measurement of the performance metrics and in the IPPM it deals with the packets for the metrics measurement. The metrics designed by the IPPM could be used by network operators and also by the end users. But in the case of BMWG, it could be used by the network operators and testing groups. Because of these reasons, the performance metrics designed by the IPPM is taken into the consideration for the designing the performance metrics in this project [53].

IPPM RFC is used to document the definition of each and every performance metrics and defines the methodology for accurately measuring and documenting these metrics. The following are the performance metrics discussed and documented in the IPPM working group.

- Bulk Transport Capacity (BTC) – RFC 3148
- One way delay – RFC 2679
- Inter packet delay variation – RFC 5481
- Packet Duplication metric – RFC 5560
- Packet loss metric – RFC 2680
- Packet reordering metric – RFC 4737
- Round trip/two-way delay metric – RFC 2681
- Round trip/two-way packet loss metric – RFC 6673

In this section, each performance metrics and the measurement of the Bulk Transport Capacity BTC, one way delay, and two delay metrics and also regarding the test equipment design mentioned in the IPPM working group will be discussed in detail.

- One-way Active Measurement Protocol (OWAMP) RFC 4656
- Two-way Active Measurement Protocol (TWAMP) RFC 5357

2.4 Bulk Transport Capacity (BTC) – RFC 3148

The Bulk Transport Capacity BTC is a measurement of a network's ability to transfer certain amount of the data from the single transport connection with the congestion awareness (e.g., TCP). The nonrational definition of the BTC is average data rate that are expected over the long term in bits per second over a single connection ideally with TCP implementation. BTC definition is generic to the entire congestion algorithm, since many congestion algorithm is allowed by the IETF community. The difference between the implementation in the congestion algorithm leads to define the transport capacity in the transport algorithm. So the definition of the generic formula for the BTC leads to the non-comparable results [39].

In the application level, the BTC of the network layer below the application level or user space is dominated by the overall elapsed time of the application by the user. According to the RFC 3148 [39], BTC is the long average data rate expressed in the bits per second over a single TCP or any congestion aware connection over the path between the source and destination. All BTC tool should report the BTC as follows

$$\text{BTC} = \frac{\text{data\_sent}}{\text{elapsed\_time}}$$

where `data_sent` represent the useful data that it means that doesn't include the header bit or a copy of it and even if the packet is retransmitted, it should be counted only once.

2.5 One way delay

The RFC 2679 [4] defines the one delay between the source hosts to destination host. The motivation for measuring the one way delay along with the two way delay has number of advantages.

- The path between source to destination host need not be the same from the destination to source. The reason is obviously due to the different sequence of routers that are between the source and destination, which will change the path between source and destination both in the forward and reverse direction. The routers use the difference forward and reverse path between the source and destination [4]
- Measuring round trip time for the asymmetric path results in the two distinct path measurement.
- Even if the path between the source and destination are symmetric, there might be difference in the distinct direction due to asymmetric queues between source host and destination host.
- For the application, the performance is mainly depends on the direction in which the data is forwarded, than the direction in which they are acknowledged [4].

The RFC 2679 definition for the one way delay is given as below,
"The one way delay dT between Src to Dst at time T, where dT is the time delay between Src sent the first bit of packet to Dst at time T and Dst received the last bit of that packet at time T+dT" [4].

- Where `T` is the time value
- `Src` is source IP address
- `Dst` is destination IP address

The value of `dT`, has to be positive value, but if the value is zero or negative, then it shows that there is a problem in the clock synchronization between the source and destination host. Testing equipment should take into account of the packet duplication, if the destination receives more copies of the data, then first data is taken into the consideration for calculating the one way delay. And also note the packet fragmentation for measuring the one way delay.

2.5.1 One way delay methodology

The following steps explains the one way delay measurement methodology

- Both the source and destination clock should be synchronized with each other.

- The source should have arrangement to send packets to destination IP address, and the destination should have arrangement to receive the packets from the source.
- Before sending the packet, the source should take a timestamp `T1`, then send the packet towards the destination.
- After getting the packet from the source at the destination within the reasonable time period. Then the destination should take the timestamp `T2`, the difference between the `T2 - T1` should give as the one way delay.

2.5.2 Errors and uncertainty

The synchronization issues between the source clock and destination clock lead to the error in the one way delay measurement. The synchronization error is termed as the `Tsynch`. If the `Tsynch` is known before the start of test, then one way delay error could be minimized. For instance, let it be assumed that source clock is ahead of destination clock by `Tsynch`. Then the error value could be minimized by adding the `Tsynch` between `T2 - T1`. The `Tsynch` in other words is the function of skew between the source and the destination clock [4].

2.5.3 Wire time vs Host timestamp

The duration of the time between the packet leaves the network interface of the source and when it arrives the network interface at the destination is defined as wire time, which will be discussed in detail in the chapter 6. The measure of time when the application in the user space in source host grabs the timestamp before sending the packet from the user space and the application in the destination grabs the timestamp after receiving the packet in the user space is defined as the host wire.

The estimation between the wire time and host time should be included in the measurement implementation. This is discussed in the results in the chapter 7. The methodology discussed in the RFC 2679 is applicable to the IP packets, for the both UDP and TCP packets.

2.6 Two way delay

The RFC 2681 [5], defines the two way delay or the round trip delay, let us discuss the motivation for the two way delay in this section from the RFC 2681.

- This metrics provides the indication of the congestion present in the path.
- The minimum value indicates the delays due to the propagation and transmission delay in the network.

- The deployment of the round trip time is easier than the one way delay, since round trip time requires only source clock for the measurement and it doesn't requires to do install measurement-specific software at the destination. This principle is used in the ICMP ping and in the well-known TCP-based methodology connectivity measure [5].

The RFC 2681 definition for the two way delay is given as below,
"The two way delay dT between Src to Dst at time T, where dT is the time delay between Src sent the first bit of packet to Dst at time T and Dst immediately send back the packet to Src, and Src receive the last bit of the same packet at T+dT" [5]
The abbreviation is the same as discussed in the section 2.5 for the one - way delay.

In the RFC 2681, even if the two way delay measurement requires only one clock source at the source, but this clock synchronization with other time servers, could cause uncertainties and error in the round trip measurement. For instance, Network Time Protocol (NTP) is used to synchronize the system clock with time servers, if the synchronization is done in between the initial timestamp and the final timestamp then it leads to the uncertainties in the round trip measurement [5].

2.6.1 Two way delay measurement methodology

The following steps explains the two way delay measurement methodology

- The source host must grab the timestamp before sending the packet to the destination IP address, there should be an information for identifying request packet from the source to the destination, and the source can identifies the response packet from the destination. This identifying information is generally placing the timestamp in the packet itself before sending it to the destination IP address.
- At the destination, the host should have the arrangement to send back the response packet to the source as soon as possible, once the packet received by the destination.
- The source hosts get the final timestamp once response packet reaches it. By subtracting, this initial timestamp and final timestamp will give us the round trip time.

Packet format by which the destination could response back to the source, is not discussed in the RFC 2681.

2.6.2 Errors and uncertainty

Similar to the one way delay, two way delays also has error and uncertainty. But when comparing to one way delay, the factors affecting the two way delay is less [5], lets discuss the error and uncertainty in this section.

- Error and uncertainty is added to source host primarily by the source clock.
- Similar to the one delay, error and uncertainty is added by the difference between the wire time vs host time.
- Processing time taken by destination to send back the response packet to the source will also add the uncertainty and error in the source host.
- Uncertainty is added by the source clock, when skew is introduced in between the initial and final timestamp between the round trip times. The problem with the two way delay is the self clock synchronization.

Error and uncertainty caused by wire time and host time is similar as discussed in the section 2.5.3

2.7 Measurement protocol

In this section, the measurement protocol to measure the performance metrics will be discussed as in IPPM working group. The RFC 4656 and RFC 5357 propose the standards for the one-way active and two-way measurement protocol for both unidirectional and bidirectional performance metrics. These RFC documents the standards for the one way delay and round trip time measurement using the standard generic testing tool.

2.7.1 One-Way Active Measurement Protocol (OWAMP) – RFC 4656

This OWAMP discuss methodology to create an environment to collect IPPM metrics from mesh of Internet paths. OWAMP consist of two protocols: OWAMP – Control and OWAMP – test. The OWAMP Control is used to initiate, start and stop the test sessions, and OWAMP test actually engage itself test data exchanges between the OWAMP servers [49]. The correlation between this methodology and flowgrind is discussed in the section 5.3.

The OWAMP – Control entity involves the session initiation, which basically exchange the source and destination address, port number between the OWAMP servers, and exchange of the testing parameter like test session length, test packet size, and it also discusses regarding the per session encryption and authentication, but these topics are out of scope of this section [49]. OWAMP divides the each entity into a logical model, and each logical model has its own standard definition and functionality. It is shown in the figure 2.1 [49].

Session Sender: The sending terminal in the OWAMP – test session.

Session Receiver: The receiving terminal in the OWAMP – test session.

Server: The node that actually involves in least one test session or more, and returns the test result to the client.

Control – Client: The node that request for the test session with servers, induces start and termination for the test session.

Fetch – Client: The entity that fetch the results from the server for the terminated test session OWAMP protocol is actually an UDP test traffic, which uses TCP connection for the OWAMP Control and UDP connection for the measurement session in the OWAMP test. The discussion regarding the implementation of the OWAMP-Control, Connection Setup and the modes of operation is out of scope for this section.

The one way delay measurement is done according to methodology in RFC 2679; the send timestamp is filled in the test data along with the error estimate, error estimate is used to share the information, regarding the clock synchronization with the UTC, using GPS hardware, or by using NTP [49]. The logical model as shown in the figure 2.1 can also be configured as client-server architecture as shown in the figure 2.2 [49].

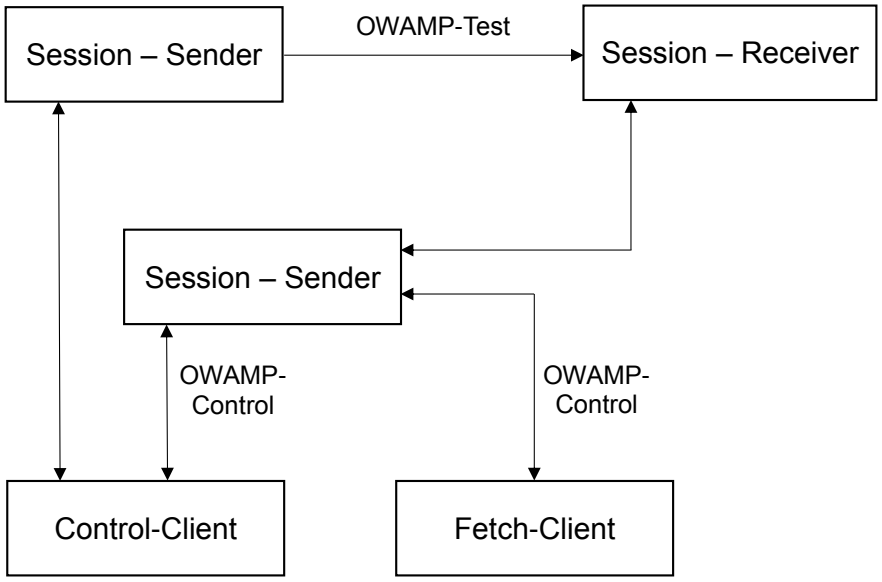

Figure 2.1: One-way active measurement protocol architecture

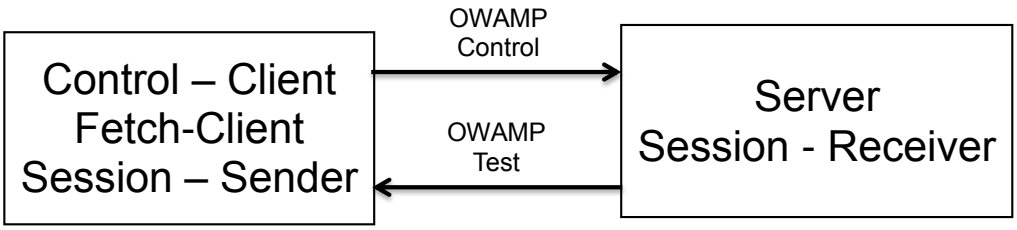

Figure 2.2: One-way active measurement protocol client and server architecture

2.7.2 Two-Way Active Measurement Protocol (TWAMP) -RFC 5357

For the two way delay measurement, there is no need for the both source and destination clock to be synchronized with each other. TWAMP implementation uses the methodology and procedure as mentioned in the OWAMP. But in addition to the OWAMP, the destination echo backs the source timestamp back to the source as response packet from the destination [35].

There are few differences between the OWAMP and TWAMP logical model, the role and function of the different logical entities are given as below. The logical model in the client - server architecture is shown in the figure 2.3.

Session-Reflector: The session receiver is called as session reflector, since session reflector has the ability to send back or echo back the packets, which it receives from the source and it doesn't collect any information regarding the packet from the server.

Server: The TWAMP server is similar to the OWAMP server but the TWAMP server doesn't have the capability to return the results because the results are not calculated at the server.

Fetch-client: This item doesn't exist in the TWAMP, since session reflector doesn't collect any information from the server. So there is no need to fetch information from the server.

The methodology to measures the two delays in TWAMP is similar to the procedure explained in the RFC 2681.

2.8 Conclusion

This chapter explains in detail, regarding the standard procedure maintained in the IETF's IPPM working group for measuring the one way delay and two delay performance metrics. This gives us the overall picture regarding the one way delay and two way

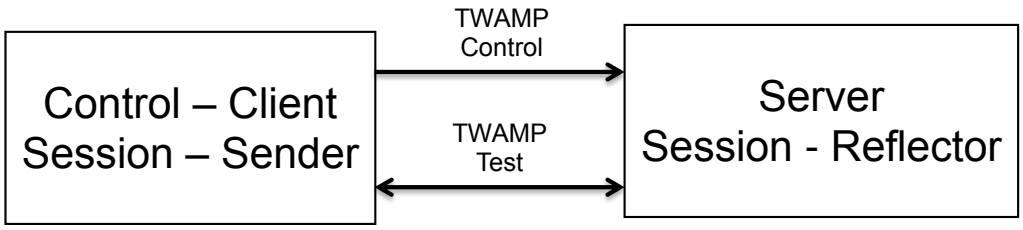

Figure 2.3: Two-way active measurement protocol client and server architecture

delay definition, methodology, errors and uncertainty in the results due to the clock synchronization issues in both source and destination side. The standard methodology to measure the latency as defined by the IETF working group and the latency measurement implementation by the performance metrics are correlated in the upcoming chapters.

3 Performance tools

With more advancement and increasing level of research in the forms of high speed networks to meet today's internet there lies a motivation to find genuine and objective benchmarks against the quality of services offered in a network. With the increase in the speed with broadband services, there is always a concern for the end users about the performance of the network.

High speed networking is not the answer in terms of performance for the broad spectrum [32]. The ability of the network to support the transactions that include the transfer of large amount of data and also support multiple and parallel transactions will give an overall picture of large network with heavy load condition and also network performance [3].

A good approach to measure the network performance is to inject test traffic into the network and measure the network performance, and then relate the performance of the test traffic to the performance of the network in carrying the normal payload. In this chapter, the performance tools and their measurement methodology will be discussed in detail.

3.1 ICMP Ping

Most widely used measurement tool is *ping*. The ping is a very simple tool to generate the Internet Control Message Protocol (ICMP) echo request packet, and directs it to destination host. When the packet is sent, the source host will start system clock timer operation. The destination host reverses the ICMP headers and sends back the ICMP echo reply packet to the source. When source receive the ICMP echo reply packet, then the timer is halted and the elapsed time is reported [32].

This indicates that the destination system is connected to the network, and is reachable from the source system. Failure of response from the destination host is not much informative, because it cannot be actually interpreted and that the destination system is not available in the network. Reason for this is that the destination response packet may have been discarded in the network due to the network congestion, and also the network path is not available to the destination. The firewall rules between the source and destination may block the destination response packet from being delivered [32].

The ICMP ping results can give some useful performance metrics information. The elapsed duration for a ping packets response, from the destination to source giving the

depiction of the response time and their standard deviation, suggests the load being experienced on the network path between the source and destination. The increased load in the network will be revealed as increased delay and their standard deviation, since the presence of large router buffers along the network path [32].

The ICMP packets could be discarded by router, if there is buffer overflows in the router. This will result in the increase of the ping loss in the network. The unpredictable high latency and loss within ping packets shows the instability within the network path. Several router architectures use fast switching paths for data transmit, however the Central Processing Unit (CPU) of the router process the ping request and response. So the ping response might be given lesser scheduling priority because router functions represent more critical operation. So in this case, there is a possibility that extended delays will be reported by the ICMP ping [32].

In more details, the typical Transmission Control Protocol (TCP) flow behavior is vulnerable to cluster into bursts of packet transmission. But ping doesn't need or not having any necessities to echo similar behavior. So the ping can only be used in a primordial way to discover the provisioned capacity of network connectivity [32].

3.2 Thrulay

Thrulay stands for THRUput and deLAY. The feature, which distinguishes the thrulay from the previous performance tools, is that thrulay reports delay information in addition to the throughput metrics. Thrulay measures throughput and round trip time by transmitting a bulk TCP stream over the network. Thrulay also supports UDP protocol in addition to the TCP, but it measures only the one-way delay for UDP packets. In UDP testing, thrulay sends a Poisson stream of very precisely positioned UDP packets [51].

The thrulay is initially developed for the measuring Round Trip Time (RTT) for the FAST TCP, but useful for standard loss – based TCP too [52]. The figure 3.1 shows the Thrulay example output for the bulk Transfer test.

3.3 TTCP and NUTTCP

The Test TCP (TTCP) [13] used to measure goodputs for both TCP and UDP packets. For the UDP packets it also displays packet loss rate. TTCP is client-server architecture, which means it can measure only unidirectional flows from client to the server [13].

The tool NUTTCP is the successor of TTCP, which has several additional features compare to the TTCP. The NUTTCP starts the test between two servers, whereas the test

```
phobos2:~/thrulay-ng-0.6.2/src% ./thrulay 172.16.121.21 -t 10s
# local window = 425984B; remote window = 425984B
# block size = 8192B
# MTU: 1500B; MSS: 1448B; Topology guess: Ethernet/PPP
# MTU = getsockopt(IP\_MTU); MSS = getsockopt(TCP\_MAXSEG)
# test duration = 10s; reporting interval = 1s
#(ID) begin, s   end, s  Mb/s    RTT, ms: min  avg     max
 ( 0)    0.000    1.000 14311.627    0.083    0.285    0.613
 ( 0)    1.000    2.000 15691.994    0.142    0.243    0.430
 ( 0)    2.000    3.000 15724.855    0.142    0.241    0.433
 ( 0)    3.000    4.000 15855.333    0.144    0.239    0.364
 ( 0)    4.000    5.000 15777.006    0.151    0.241    0.426
 ( 0)    5.000    6.000 15895.364    0.151    0.239    0.368
 ( 0)    6.000    7.000 15860.941    0.145    0.239    0.361
 ( 0)    7.000    8.000 15791.112    0.151    0.241    0.373
 ( 0)    8.000    9.000 15908.691    0.151    0.238    0.426
 ( 0)    9.000   10.000 15872.557    0.150    0.240    0.361
#( 0)    0.000   10.000 15668.922    0.083    0.244    0.613
#(**)    0.000   10.000 15668.922    0.083    0.244    0.613
```

Figure 3.1: Thrulay example output: TCP bulk transfer test

started by NUTTCP client could be from the system. The NUTTCP client submit the test parameters to servers and then servers will establish the data connection between them to perform the measurements [42]. The test results are aggregated and sent back to the NUTTCP client, where it is displayed. The figure 3.2 shows the NUTTCP example output for the bulk Transfer test.

The NUTTCP report the total amount of data sent, average throughput per second, CPU usage in percentage for both source and destination, uses the `TCP_INFO` socket option to display the number of retransmission on NUTTCP data connection between the servers, and average RTT measured. NUTTCP also reports the results per interval.

3.4 Iperf

Iperf is relatively simple tool, it is based on the client-server architecture. It is primarily used to measure the goodput. To start the test, the Iperf client connects to the Iperf server and sends the test parameter to the server and then starts the bulk data transfer between client and server. The Iperf measures both the TCP and User Datagram Protocol (UDP) bulk transfer test. By optional in the TCP bulk transfer test, data can optionally be sent in parallel with multiple connections to the same server via multiple threads. The Intermediate tests are displayed at the both Iperf server and client according to the configured time interval testing. The UDP bulk transfer test can also measure the datagram loss rate and delay jitter [33].

But in addition to these advantages, the Iperf also have few demerits. Iperf can only test against one server at a time and unlike NUTTCP does the Iperf does not support third party tests. The Iperf can measure the parameter unidirectional from server to client

```
phobos1:~/nuttcp-5.5.5% nuttcp -v -v -i1 172.16.121.22
nuttcp-t: v5.5.5: socket
nuttcp-t: buflen=65536, nstream=1, port=5001 tcp -> 172.16.121.22
nuttcp-t: time limit = 10.00 seconds
nuttcp-t: connect to 172.16.121.22 with mss=1448
nuttcp-t: send window size = 23040, receive window size = 178560
nuttcp-r: v5.5.5: socket
nuttcp-r: buflen=65536, nstream=1, port=5001 tcp
nuttcp-r: interval reporting every 1.00 second
nuttcp-r: accept from 172.16.121.21
nuttcp-r: send window size = 23040, receive window size = 179520
 2580.6250 MB /   1.00 sec = 21646.5311 Mbps
 2635.6250 MB /   1.00 sec = 22110.5074 Mbps
 2637.6250 MB /   1.00 sec = 22125.5597 Mbps
 2641.1875 MB /   1.00 sec = 22156.3519 Mbps
 2640.3125 MB /   1.00 sec = 22148.3029 Mbps
 2646.1250 MB /   1.00 sec = 22196.9724 Mbps
 2643.8750 MB /   1.00 sec = 22178.9855 Mbps
 2644.5000 MB /   1.00 sec = 22183.6739 Mbps
 2644.6250 MB /   1.00 sec = 22183.7907 Mbps
 2645.6875 MB /   1.00 sec = 22194.3677 Mbps
nuttcp-t: 26366.3125 MB in 10.00 real seconds = 2699908.24 KB/sec = 22117.6483 Mbps
nuttcp-t: 421861 I/O calls, msec/call = 0.02, calls/sec = 42186.07
nuttcp-t: 0.1user 4.8sys 0:10real 49% 0i+0d 1118maxrss 0+0pf 142104+37csw

nuttcp-r: 26366.3125 MB in 10.00 real seconds = 2699623.97 KB/sec = 22115.3196 Mbps
nuttcp-r: 421864 I/O calls, msec/call = 0.02, calls/sec = 42181.92
nuttcp-r: 0.0user 9.9sys 0:10real 99% 0i+0d 96maxrss 0+16pf 4+105csw
```

Figure 3.2: NUTTCP example output: TCP bulk transfer test

and it doesn't support bidirectional TCP and UDP goodput. In order to simulate the bidirectional TCP connection, the two unidirectional connections are used in the parallel. The figure 3.3 shows the Iperf example output for the bulk Transfer test.

3.5 Netperf

Netperf is also a client-server based measurement tool, similar to Iperf and doesn't support the third party test. It can also only test against one single server at a time. Netperf is able to use Unix Domain sockets and the data link provider interface and supports Stream Control Transmission Protocol (SCTP) in addition to the TCP and UDP protocol. It is developed by Hewlett-Packard. Netperf provides different testing for the different protocol. TCP_Steam is the most basic test for the TCP protocol, which used to measure unidirectional TCP good put for the bulk data transfers. The figure 3.4 shows the Netperf example output for the bulk Transfer test.

Netperf also supports the request-response test called as TCP_RR in which Netperf measure the transaction rate [41]. In addition to these tests, Netperf is used to measure TCP connection establishment and also the closure. If these tests are combined with the request-response (TCP_RR), the resulting network load is not equivalent to the Hypertext

```
phobos1:~/iperf/src% iperf3 -c 172.16.121.22 -B 172.16.121.21 -l 8192 -i 1
Connecting to host 172.16.121.22, port 5201
[  4] local 172.16.121.21 port 36904 connected to 172.16.121.22 port 5201
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-1.00   sec  1.91 GBytes  16.4 Gbits/sec    0
[  4]   1.00-2.00   sec  1.91 GBytes  16.4 Gbits/sec    0
[  4]   2.00-3.00   sec  1.91 GBytes  16.4 Gbits/sec    0
[  4]   3.00-4.00   sec  1.92 GBytes  16.5 Gbits/sec    0
[  4]   4.00-5.00   sec  1.91 GBytes  16.4 Gbits/sec    0
[  4]   5.00-6.00   sec  1.92 GBytes  16.5 Gbits/sec    0
[  4]   6.00-7.00   sec  1.92 GBytes  16.5 Gbits/sec    0
[  4]   7.00-8.00   sec  1.92 GBytes  16.5 Gbits/sec    0
[  4]   8.00-9.00   sec  1.92 GBytes  16.5 Gbits/sec    0
[  4]   9.00-10.00  sec  1.92 GBytes  16.5 Gbits/sec    0
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-10.00  sec  19.2 GBytes  16.5 Gbits/sec    0         sender
[  4]   0.00-10.00  sec  19.2 GBytes  16.5 Gbits/sec              receiver

iperf Done.
```

Figure 3.3: Iperf example output: TCP bulk transfer test

Transfer Protocol (HTTP) traffic. The results in such scenarios could be considered as basic traffic generation test. The Netperf can measure the overall CPU utilization on both source and destination host as well as the service demand, which is the number of CPU time spent per measurement unit, for example, the CPU time needed for a single transaction for the request- response tests.

```
phobos1:~/netperf/src% ./netperf -L 172.16.121.21 -H 172.16.121.22 -p 4500 -v 2 -- -M
    8192 -m 8192
MIGRATED TCP STREAM TEST from 172.16.121.21 () port 0 AF_INET to 172.16.121.22 () port
    0 AF_INET : interval
Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    10^6bits/sec

 87380  16384   8192    10.00    12704.16

Alignment      Offset         Bytes    Bytes       Sends   Bytes    Recvs
Local  Remote  Local  Remote  Xfered   Per                 Per
Send   Recv    Send   Recv             Send (avg)          Recv (avg)
    8       8       0       0 15881691136  8192.00    1938683   8192.00 1938684

Maximum
Segment
Size (bytes)
  1448
```

Figure 3.4: Netperf example output: TCP bulk transfer test

3.6 Drawback of existing performance tools

The main drawback of previously discussed performance measurement tools is their client-server architecture, which makes the transmission of test data through the network along multiple paths in parallel difficult. To test the multiple paths, the clients need to execute in parallel on different nodes and have to run the server multiple times as well. Using the client – server architecture based tools to set up testing framework that involves multiple clients in a monotonous task. As well as these scenarios lead to the synchronization problems as well. For example, even if the two clients start the testing at the same time, there might be a chance that one client is still exchanging the testing parameters with its server, while the other clients have already begun the actual data testing with the server. The consequence of this lack of synchronization leads to the inaccurate results in the testing [62].

The bidirectional protocols like TCP can send the data in the both directions at the same time. But unfortunately all these tools support only unidirectional data transfer that means these tools can only generate unidirectional traffic on an individual test connection. Trying to simulate the bidirectional loads through the use of two parallel unidirectional test connections doesn't represent the accurate model of true bidirectional test connections [62].

Table 3.1: Performance measurement tools feature matrix

Feature	nuttcp	iperf	thrulay	netperf
TCP	✓	✓	✓	✓
UDP	✓	✓	✓	✓
IPv6	✓	✓	✓	✓
RTT	✓	-	✓	-
IAT	✓	-	✓	-
Network Transactions/s	-	-	-	✓
CPU utilization	✓	-	-	✓
third party tests	✓	-	-	-
interval reports	✓	✓	✓	✓
scheduling	-	-	-	-
control/test interface separated	✓	-	-	-
bi-directional traffic	✓	✓(pseudo)	-	-
select congestion control	✓	✓	-	-
TCP_INFO X(partial)	✓(partial)	-	-	-
Rate Limiting	-	✓	UDP only	✓
Request Response	-	-	-	✓(basic)

3.7 Conclusion

This chapter discuss in details regarding the present performance measurement tools and the comparison regarding their features is shown in the table 3.1. All the tools discussed in this chapter couldn't generate the realistic internet traffic. This is considered as the one of the disadvantage of these tools. The details regarding the traffic generation is discussed in the next chapter.

4 Traffic Generation model

This chapter gives an overview regarding the stochastic traffic generation in the flowgrind. Additionally this section gives the mathematical model behind the stochastic traffic generation in basic. The section gives the details regarding the traffic model and scenarios generated by traffic generation.

4.1 Introduction to Traffic Generation model

Developing a traffic generation, for example a pragmatic internet is a challenge task. In general there are two primitive methods to generate manifold internet traffic. A basic and easy approach is to record cluster of traffic, and subsequently re-run them. This approach is called *traffic replay* [31]. Another approach to generate pragmatic traffic by using the stochastic process, this method is called as the *stochastic traffic generation* [47]. Both the methods, has its own advantages and disadvantages, because of their way, in which these approaches are developed. The traffic replay method is more pragmatic, since trace record is the combination of the data streams and the traffic attributes and it reproduces the trace records with the same traffic attributes [31]. Despite the easy implementation of the traffic replay, it has its own demerits because traffic replay considers the trace records as the black box and it is difficult to change the traffic to test different test scenarios [47]. And other demerits of the traffic replay approach are that it requires additional procedure and resources to record the traces, process and store them.

In contrast, the stochastic traffic generation generates the traffic based on the mathematical model for different traffic and workload characteristics. One of the useful features in the stochastic model is that the parameters can be extracted from quantitative analysis. This feature in the stochastic traffic generation compels it to do experiment protocols and evaluate them, before actually doing the implementation work for the protocols [47]. Another disadvantage of the traffic replay method is that difference between the traffic noise and the payload in the trace records requires additional information for boundaries in the data set [47]. The whole trace records are needed to store in traffic replay, but in contrast to it, the stochastic traffic generation, requires to store only the parameter to generate the stochastic traffic generation and random state number.

The flowgrind stochastic traffic generation has been discussed in this section 5.8

4.2 Mathematical Background

The branches of mathematics namely probability theory and statistics are the bases for the stochastic. The probability distribution plays a vital role in the probability theory. The random variable is the basis for the probability distribution. For each experiment, the random element allocates a value to each possible result [10].

For instance, let us take a scenario of random variable Y for a coin flipping, which could be like this

$$Y = \begin{cases} 0, & \text{if head,} \\ 1, & \text{if tail.} \end{cases}$$

In the following procedure, let us add the probability to the value of possible results which leads to the probability distribution. In our experiment of coin flipping, the value of the probability for the head and tail should be 0.5. But let us consider that the coin is biased or manipulated and the final results could be possible as.

$$Y = \begin{cases} 1/3, & 0, \text{if head,} \\ 2/3, & 1, \text{if tail.} \end{cases}$$

The given resultant value is called as the discrete probability distribution. If there is a finite set in the probability distribution, where the sum of the set is 1, then that probability distribution is considered as the discrete [10].

The continuous probability distribution in contrast to discrete probability distribution has a continuous random element Y with a probability density function $f(Y)$:

$$\Pr[a \leq Y \leq b] = \int_a^b f(y)dy$$

A probability density function for the random element Y plots the corresponding possibility for this element to exist at a given point in a set. The integral of random element density result in the probability for the random element within the given set. This statement implicit the probability of single value is zero, for instance for the z value, the probability $z \leq Y \leq z$ is zero. This is because the integral value with same upper and lower bound is always zero [10].

4.3 Traffic model

This section contains information regarding the cdma 2000 Evaluation methodology theoretical mode [1] for the internet traffic, mainly for the HTTP protocol.

The reference to the technical document, which describes the traffic model for the different network traffic, is designed on the concepts of empirical analysis of cellular network traffic [1]. Although this model is recommended for the traffic model for the 3G hardware evaluation, the basic traffic model for the Hypertext Transfer Protocol (HTTP) traffic can be adopted for other protocol evaluation as well. From the technical document [1], the HTTP traffic has six components. The main object size (S_M), the embedded object size (S_E), the number of embedded objects per page (N_d), the reading time (D_{pc}), the initial reading time (D_{ipc}) and the parsing time (T_p) taken from the technical document [1]. The HTTP request size is fixed for 350 byte in this model.

The requirement for the traffic model is simple. The implementation of passing parameter should be simple and at the same time, it should be able to emulate protocol, for instance like the HTTP traffic and Teletype Network (TELNET). Since the cdma 2000 evaluation methodology uses more parameter, it should be designed with less parameter. The chosen parameter for the traffic models are given below.

Request size: This size represents the single request block. For example, in a single website the block represent the smallest unit of the transferred data. It is denotes as S_q

Response size: This size represents the single response block. If the response block is greater than zero, then with each request block, a response block is requested according to the response size. It is denoted as S_p

Interdeparture time: The time gap between two request size block can be used for the two purposes, one is for the rate limit request, in the case of reading a website in HTTP protocol and another one is to achieve the wait time, in the case of waiting for the user input in the TELNET protocol. It also represents the Interdepature Gap. It is denoted as T_g

The parameter is used in the cdma 2000 Evaluation methodology. For instance, the parsing time (T_p), reading time (D_{pc}), initial reading time (D_{ipc}) can be managed by using only the Interdepature time T_g

4.4 Traffic generation use cases

This section generally list the possible and general use cases possible through the Traffic model, this gives a general overview picture of the use cases used in the actual measurement scenario and discussed in detail in result section 7.3

Rate limited flows: This scenario is used in generation use case for rate timed media streaming and sender limited flows. The interdeparture T_g is added between two request block. The sending rate can be expressed as *byte/s* , or in the *block/s* format. The interpacket gap can be computed from specifying the target sending rate.

$$\text{Interdepature time } Tg\,[s] = \frac{block\ size\,[bytes]}{write\ rate\left[\frac{bytes}{s}\right]}$$

The burst behavior in the Transmission Control Protocol (TCP) can be affected by changing the written or request block size from the above formula. It always writes the whole data into socket at a single shot, this leads to a smooth data transfer, whether the block size is of greater or smaller value. This transfer characterizes could be made less deterministic, by applying the normal distribution for the interpacket gap.

Bulk transfer flow: The bulk transfer flow can be modelled using the traffic generation model with constant distribution for the request size S_q, and both respond size S_p and interdeparture time T_g are set to zero. For instance, File Transfer Protocol (FTP) the file transfer protocol could be emulated by the bulk transfer flow.

Request-Response flow: The common flow for many existing traffic model is the request-response flow. The simple concept of the request and response flow is that a sender sends a block (the request size block) and receiver responds back to the sender with another block(the response block). This communication model can be emulated by changing the request size S_q and response block S_p. And the interdeparture time T_g with different values, can emulate different protocols like HTTP, TELNET and Simple Mail Transfer Protocol (SMTP).

4.5 Conclusion

This chapter gives the over views of the fundamental and basic procedure to implement the generic traffic model and in addition it also explains actual implementation of the Stochastic model in the flowgrind. The practical use of different use cases for the measurement results are discussed in this section and these are directly applied to the test procedure. In order to understand the flowgrind traffic generation in the section 5.8 and the test procedure in the section 7.3 this chapter is useful.

5 Flowgrind

The objective of the thesis is to develop the latency measurement module using the existing and suitable performance tools. The chapter 2 discusses regarding the methodology of latency measurement procedure, and chapter 4 discusses regarding the generation of diverse traffic to emulate the workload condition. This chapter gives details regarding performance measurement tool *flowgrind*, which supports the traffic generation and also discuss the advantage of implementation of the latency measurement module in the flowgrind.

5.1 Introduction to flowgrind

Flowgrind supports distributed architecture and performs measurements against an arbitrary number of end-points simultaneously. One or more test connections are called as *"flows"* in the flowgrind. These flows could be scheduled to run consecutively, interleaved and fully synchronized. Each flow could be assigned with a duration and an initial delay. Actual test connection starts after the initial delay. In addition to this, each flow have its own testing parameters, which are called as *"flow options"*, which can be set individually for each direction [62].

The flowgrind continuously outputs the data according to the flow option interval timing, which can be configured individually with millisecond precision. This provides not only very fine grained reporting, but also coarse intervals like reporting in seconds and minutes. Flowgrind have an extra feature to report regarding the Transmission Control Protocol (TCP)-specific performance metrics. These metrics are obtained from the linux kernel using the `TCP_INFO` socket option. Similar to all other performance metrics, TCP-specific metrics are collected by the flowgrind and sample at the end of reporting time interval specified by its flow option. The figure 5.1 and 5.2 shows the basic flowgrind output

5.2 History

The flowgrind performance tool is loosely related to thrulay tool developed by Stanislav Shalunow [50] which has been developed back 2005. These inspired researchers [62] developed a performance measurement tool specialized for evaluation of TCP/IP stack and measuring the performance metrics. The earlier version of the flowgrind aimed to

```
phobos2:~/flowgrind% ./flowgrind -c through,transac,interval,iat,blocks -i1 -T s=3 -A s -H s=172.16.121.22,d
    =172.16.121.21
# Date: 2015-04-26-21:42:25, controlling host = phobos2.mgmt.muclab, number of flows = 1, reporting interval = 1.00s
    , [through] = 10**6 bit/second (flowgrind-0.7.5-124-gfd4c00)
# ID  begin     end       through      transac     requ    resp  min RTT  avg RTT  max RTT  min IAT  avg IAT  max IAT
#       [s]       [s]      [Mbit/s]     [#/s]       [#]     [#]     [ms]     [ms]     [ms]     [ms]     [ms]     [ms]
S  0   0.000     1.000  10833.598809  164564.66    165321       0    0.272    4.473    5.079    0.272      inf      inf
# ID  begin     end       through      transac     requ    resp  min RTT  avg RTT  max RTT  min IAT  avg IAT  max IAT
#       [s]       [s]      [Mbit/s]     [#/s]       [#]     [#]     [ms]     [ms]     [ms]     [ms]     [ms]     [ms]
D  0   0.000     1.000     52.685356      0.00         0  164642      inf      inf      inf      inf    0.006    0.118
D  0   1.000     2.000     54.543320      0.00         0  170456      inf      inf      inf      inf    0.006    0.070
S  0   1.000     2.000  11171.223886  170459.35    170448       0    4.199    4.341    4.513    4.199      inf      inf
D  0   2.000     3.000     54.608431      0.00         0  170642      inf      inf      inf      inf    0.006    0.070
S  0   2.000     3.000  11182.850713  170644.76    170633       0    4.211    4.336    4.515    4.211      inf      inf

# ID   0 S: 172.16.121.22 (Linux 3.19.0.muclab+), random seed: 2373624646, sbuf = 16384/0 [B] (real/req), rbuf =
    87380/0 [B] (real/req), SMSS = 1448 [B], PMTU = 1500 [B], Interface MTU = 1500 (Ethernet/PPP) [B], duration =
    3.000/3.000 [s] (real/req), through = 11062.544448/53.937930 [Mbit/s] (out/in), transactions/s = 168556.03
    [#], request blocks = 506402/0 [#] (out/in), response blocks = 0/505667 [#] (out/in), RTT = 0.272/4.382/5.079
    [ms] (min/avg/max)
# ID   0 D: 172.16.121.21 (Linux 3.19.0.muclab+), random seed: 2373624646, sbuf = 16384/0 [B] (real/req), rbuf =
    87380/0 [B] (real/req), SMSS = 1448 [B], PMTU = 1500 [B], Interface MTU = 1500 (Ethernet/PPP) [B], through =
    53.945681/11048.075463 [Mbit/s] (out/in), request blocks = 0/505740 [#] (out/in), response blocks = 505740/0
    [#] (out/in), IAT = 0.003/0.006/0.118 [ms] (min/avg/max), delay = 0.192/4.264/4.944 [ms] (min/avg/max)
```

Figure 5.1: Flowgrind example output: Measurement without kernel output

```
phobos2:~/flowgrind% ./flowgrind -c through,kernel, -i1 -T s=3 -A s -H s=172.16.121.22,d=172.16.121.21
# Date: 2015-04-26-21:46:29, controlling host = phobos2.mgmt.muclab, number of flows = 1, reporting interval = 1.00s
    , [through] = 10**6 bit/second (flowgrind-0.7.5-124-gfd4c00)
# ID       through  min RTT  avg RTT  max RTT  cwnd  ssth  uack  sack  lost  retr  tret  fack  reor  bkof   rtt
    rttvar     rto  ca state  smss  pmtu
#         [Mbit/s]     [ms]     [ms]     [ms]   [#]   [#]   [#]   [#]   [#]   [#]   [#]   [#]   [#]   [#]  [ms]  [
    ms]    [ms]                [B]   [B]
S  0  10967.540876    0.220    4.394    4.982   204   171   136     0     0     0     0     0     3     0   0.1
    0.0   201.0     open  1448  1500
S  0  11160.996149    4.180    4.326    4.499   204   171    34     0     0     0     0     0     3     0   0.1
    0.0   201.0     open  1448  1500
D  0     53.336954      inf      inf      inf    10    17     1     0     0     0     0     0     3     0   0.1
    0.0   201.0     open  1448  1500
D  0     54.499789      inf      inf      inf    10    17     1     0     0     0     0     0     3     0   0.1
    0.0   201.0     open  1448  1500
S  0  11174.307930    4.153    4.320    4.499   204   171     9     0     0     0     0     0     3     0   0.1
    0.0   201.0     open  1448  1500
D  0     54.573222      inf      inf      inf    10    17     1     0     0     0     0     0     3     0   0.1
    0.0   201.0     open  1448  1500

# ID   0 S: 172.16.121.22 (Linux 3.19.0.muclab+), random seed: 3826289759, sbuf = 16384/0 [B] (real/req), rbuf =
    87380/0 [B] (real/req), SMSS = 1448 [B], PMTU = 1500 [B], Interface MTU = 1500 (Ethernet/PPP) [B], duration =
    3.000/3.000 [s] (real/req), through = 11100.950719/54.125035 [Mbit/s] (out/in), transactions/s = 169140.73
    [#], request blocks = 508165/0 [#] (out/in), response blocks = 0/507426 [#] (out/in), RTT = 0.220/4.346/4.982
    [ms] (min/avg/max)
# ID   0 D: 172.16.121.21 (Linux 3.19.0.muclab+), random seed: 3826289759, sbuf = 16384/0 [B] (real/req), rbuf =
    87380/0 [B] (real/req), SMSS = 1448 [B], PMTU = 1500 [B], Interface MTU = 1500 (Ethernet/PPP) [B], through =
    54.136620/11087.179770 [Mbit/s] (out/in), request blocks = 0/507530 [#] (out/in), response blocks = 507530/0
    [#] (out/in), IAT = 0.003/0.006/0.085 [ms] (min/avg/max), delay = 0.021/4.224/4.848 [ms] (min/avg/max)
```

Figure 5.2: Flowgrind example output: Measurement with kernel output

evaluate the TCP in Wireless Mesh Network (WMN), but it was found out that, at that time there was no appropriate tool available for evaluating the TCP for the WMN. So the flowgrind performance tool was developed based on the inspiration and concepts of the thrulay. In the later release, distributed architecture feature was added to the flowgrind to overreach the distributed architecture issue related with the WMNs. Now the flowgrind is a completely independent performance tool, which has improved a lot when compared to the thrulay performance metrics. Unlike thrulay measure, it is used to measure both the bulk traffic, request-respone test, show the TCP-specific information,

latency measurement. Next logical steps towards the flowgrind is to improve the performance and stability in the tool [62].

5.3 Flowgrind architecture

As mentioned earlier, flowgrind supports distributed architecture. This is because the flowgrind by itself is splitted into two parts, the flowgrind daemon (flowgrind) and flowgrind controller. The flowgrind controller doesn't take part in actual testing and measurement. It is rather used to pass the flow options and flow testing parameter to daemons which are running between the two servers and then these daemons will actually start the testing and measurement process. The performance metrics are sampled by daemons running on the servers. The controller will contact the daemons at the end of every reporting interval. Then the daemons will send back the collected performance metrics data to the controller, and controller displays the results. So the controller doesn't need to be a part of the tested network. So it is possible to conduct testing between arbitrary servers running flowgrind daemons [62]. So the flowgrind architecture can be correlated with One-way Active Measurement Protocol (OWAMP) in the subsection 2.7.1 and with Two-way Active Measurement Protocol (TWAMP) in subsection 2.7.2 and the logical model for the flowgrind architecture can be drawn as shown in the figure 5.3 based on the logical model discuss in these subsection.

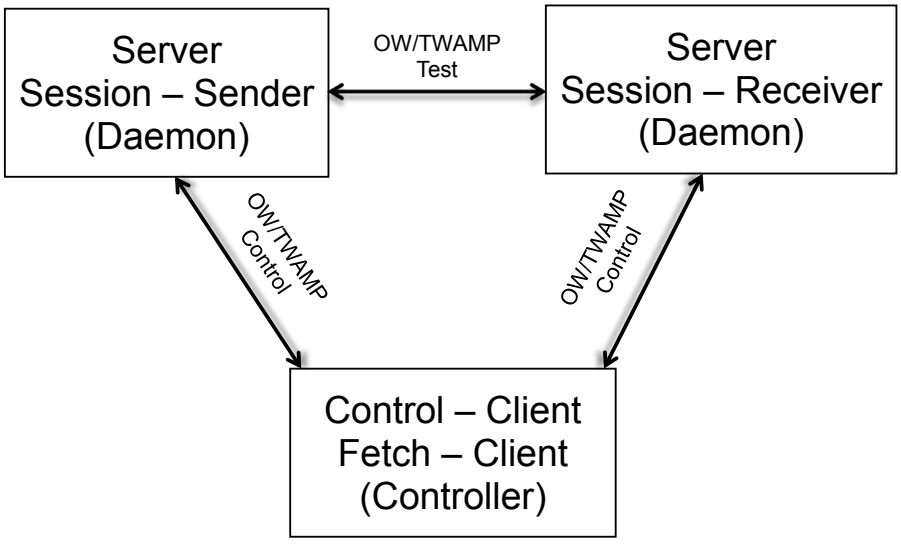

Figure 5.3: Flowgrind architecture

5.4 Flowgrind interprocess communication

The Interprocess communication between the daemon and controller is realized by the Remote Procedure Call (Remote Procedure Call (RPC)). As the name suggests RPC facilitates one process to execute a function in another process and the returned values are transferred back to the caller. Flowgrind employs Extensible Markup Language Remote Procedure Call (XML-RPC) [24] for their inter-process communication between the daemon and controller. It uses Extensive Markup Language (XML) [30] to encode its data by serializing it, which makes implementation, handling and debugging easy tasks and Hypertext Transfer Protocol (HTTP) [55] as a transport mechanism. Hence the flowgrind controller don't even need to run on the same machine, with the daemon, as standards for handling RPC over network connections exist.

The library used by the flowgrind is XML-RPC for C+/C++ (XML-RPC), which offers functionality to transport XML-RPC data and provide an interface between daemon and controller to construct and parse the XML encoded messages. But it is possible to communicate with daemon from another client other than flowgrind controller through XML-RPC communication.

XML-RPC makes it possible to create flows between any arbitrary nodes running the daemon. The flowgrind controller processes the flow options and testing parameter and transmits these data to daemons using RPC and as well the daemons will send back the test results using RPC. So that the results get aggregated in one location in the controller for the further analysis. Flowgrind daemons have flow option to set XML-RPC to bind to different network interface, so they do not influence the actual test connection between the testing daemons. This is essential factor for the tests which involves with relatively low bandwidth as the fraction of control data gets higher and the lower the overall bandwidth [62].

5.5 Command line arguments

The flowgrind controller supports a large number of flow configuration options. As discussed in the earlier section, the flowgrind supports the bidirectional flows, and the flow configuration can be applied individually to each daemon (referred as flow endpoint). Beside the flow options, the flowgrind has general options as shown in the figure 5.5, which applies to all the flows in the controller. General option includes the number of flows, displaying particular performance metrics and report intervals. The flow option as shown in the figure 5.6 includes test option for each flow. For example to set block (message) size for each endpoint: - S x=#, the flowgrind endpoints are denoted by 'x'. The x need to be replaced with either 's' for the source daemon and 'd' for the destination daemon or 'b' for both source and destination daemon. For example to set source and destination message block size, the corresponding flow option is -S s=4096, d=512. By default all the flow options are applied to all flows. Using -F # , #* flow option,

a particular flows can be configured. For instance -F 1, 4, 6 sets all further flow options up to the next -F would apply to flows 1, 4 and 6 flow IDs and flow ID numbering starts with 0 [62].

To set the flowgrind endpoint host, the -H option is used. In addition to the address of the test host it is also possible to give an alternative RPC address for the controller to connect to daemon for each flowgrind endpoints by using the option -H x=HOST[/CONTROL[:PORT]], for example

-H s=192.168.10.1/host1:9000,d=192.168.11.1/host2:9000

would set up a flow connection or control connection between 192.168.10.1 interface in the host1 and 192.168.11.1 interface in the host2 at the port number 9000. The controller can address the daemon machine using the Domain Name System (DNS) names host1 and host2 respectively [62].

Another interesting flowgrind flow options are TCP socket options (refer the figure 5.7), -O x=OPT, where the OPT are the test socket option. For example, `TCP_NODELAY` disable Nagle algorithm on test socket. This option allows the flowgrind to check with various socket options. The flow duration and initial delay before the host starts to send are set using the option -T and -Y. The flow options can be applied to the individual endpoints as similar to the message size example shown above.

When the test runs, the flowgrind controller gets the interval reports from the flowgrind daemons and prints the report lines as shown in the figures 5.1 and 5.2. The flowgrind doesn't make any guarantees on the order that the reports from different flows or endpoint get printed. Only the reports from the same flow and endpoint are shown in the order. The reason is that gathering and displaying the results are done asynchronously. The needs are to be taken into consideration when parsing the output.

The header with performance metrics named in the column and units are displayed periodically. The column width will be resized dynamically according to the use of the available space. The columns can be disabled according to the configuration option. Final results are displayed after the test duration is completed or if the test is finish forcefully interrupted. Final test report contains more information regarding send buffer and receive buffer size and also all the cumulated results from the daemon.

#: Represents the flow endpoint, either S for sender or R for receiver.

ID: The flow Identifier, if the number of flows are 3, then ID values 0, 1, and 2 represents the flow ID of source and destination.

begin /end : Represent the time the daemon begin and end the results for the interval duration.

through: The throughput of the flow endpoint during this report interval.

transac: The Network Transaction per second.

requ/ resp: The number of successful written request and response blocks in the report interval. This column is not shown by default because it is the similar to the transac column.

IAT/RTT: The application level block inter-arrival time and Application level block Round Trip Time (RTT). For both values the minimum, average and maximum encountered values in that interval are displayed in addition to the arithmetic mean. The flowgrind displays "inf", if no corresponding block arrives during the report interval [62].

The following values are extracted from the kernel from the socket option `TCP_INFO` at the end of every reporting interval and represent the internal state of the Linux TCP stack; according these are only available in Linux. Also the meaning can differ in newer versions of Linux.

cwnd: Size of TCP congestion window (in number of segments).

ssth: The slow start threshold of the sender in the number of segments. Arbitrary numbers are optionally replaced by their symbolic name to enhanced readability. Linux for example initializes the `SSTHRESH` with `INT_MAX (2147483647)`.

uack/sack: Number of unacknowledged and selectively acknowledged segments.

lost: Number of segments assumed lost at the end of the report interval.

retr/ tret: This represents the number of all retransmission (retr) and timeout based retransmission (tret).

fack: Number of segments between the highest selectively acknowledged sequence number and Send Unacknowledged.

reor: Segment reordering metric. The Linux kernel can detect the reordering and cope up without loss of performance, whenever the distance a segment gets preempted does not exceed the reordering measurement [30].

bkof: Number of consecutive exponential back offs in the current recovery phase.

rtt/rttvar: TCP round trip time estimation and its variation in the millisecond (ms)

rto: The TCP retransmission timeout is given in the ms

ca state: Internal state of the congestion control state machine as implemented in the linux kernel. The ca state is as follows open, disorder, cwr, recovery or loss [30].

- **Open:** It is the normal state. This statement shows that there are no issues with the TCP connection
- **Disorder:** This status is similar to the Open, but TCP enter this status upon obtaining duplicate ACKs or selective acknowledgements

CWR: TCP enters this status when the size of the congestion windows get lowered due to the receiving Internet Control Message Protocol (ICMP) Source Quench message or a notification from Explicit Congestion Notification (ECN).

Recovery: This status represents that TCP is in the fast-retransmitted and that the congestion window get lowered.

Loss: This TCP status shows that the Retransmission Timeout (RTO) expires. Similar to Recovery state this represents that the congestion window got lowered in this state.

smss: This performance metrics represent the number of segments, and this shows the sender maximum segment size that is the size of the largest segment the sender can transmit.

pmtu: This performance metrics represent the Path Maximum Transmission Unit (MTU), the MTU along the path between the source and destination in the flow.

After the interval test results, a final report is displayed for each flow end-points with their flow ID. The final report consists of the flow endpoint server name, OS, size of the send and receives buffer and the initial advertised window. Flowgrind try to guess the network type by using the value of the Maximum Segment Size (MSS) and MTU, if it couldn't find the network used, then it displays `UNKNOWN`. After that the initial delay for the flow is displayed, in the case that is given through the controller and as well as the flow duration for each flow endpoint.

Afterwards, the average goodputs and number of block will be sent and then received by the flowgrind. Following the final report, the flowgrind controller closes its log file, kill and tear down its RPC client.

5.6 Traffic dumping

Flowgrind is not only for analysis the TCP, and measuring the performance metrics by doing the bulk transfer and request-response test but it can also be used to analyze the a qualitative analysis of the behavior of the TCP stack. For incorporating qualitative analysis of TCP, flowgrind gathers the needed information by recording the whole TCP connection. Such recording is usually called as dump.

A famous tool to perform this task is tcpdump [59], which is developed on the top of the libcap [17]. Even the usage of the tcpdump is easy, but the scheduling and multiplexing different flows along with flowgrind is not easy. To facilitate this process, flowgrind is supported with extension to support the automatic dump for each flow. Flowgrind by itself use the libcap to regenerate the dump automatically, which produces the same output format similar to the tcpdump.

The dump request for the particular flow endpoint daemon is done by using the flowgrind controller. Then flowgrind daemon creates an auxiliary thread per flow which dumps the TCP information with respective data test connection in the flow it maintains. Once the data test duration is finished then the corresponding thread for the dump is destroyed and then all additionally allocated resources are freed. The dump extension is supported to the both source and destination, and also could be on the both sides. For the bulk transfer, running dump extension in the source side make more sense because it is active part where behavior has to be analyzed. But in the case request and response test, depending upon the source request and destination response, it is always good to extend the dump extension in both source and destination side.

5.7 Flow scheduling

Comparing with Netperf performance tool, flowgrind doesn't use multiple process nor threads to handle multiple and parallel, and concurrent data test connection between daemons. The flowgrind daemon actually multiplexes all the flows into a single thread. Maintaining all the flows in the single thread make sense because most of the performance tools are I/O bound and most of the time, they wait for the network. Test connection with multiple flows with separate thread and process scheduler of the OS can lead to inaccurate results with multiple flows handled by different thread [62]. The inaccuracy in the results occurs because different threads are often scheduled and implicitly gets different shares of the possible bandwidth. By using single thread for processing flows, flowgrind avoids the plausible unfairness issues caused by the task scheduler of the OS [62].

Flowgrind basically iterates through non-blocking select-loop over the socket file descriptors of all active flows, processing the file descriptors which are ready for read and respectively for write. Introducing the scheduling in flowgrind might lead to possible inaccuracy of test duration and the reporting intervals. For example, when thread involves in read or writes operation, then the report thread might not able to operate and this leads to miss its report interval. These cases arise especially during large block messages for example 1MB, since read and write operations will take a significant time. Flowgrind checks the timing before the read or write operation instead of planned report interval using scheduling. Control operation using XML-RPC is done by separate auxiliary threads, to avoid it influencing the actual data test connection between the daemons. Processing both the data test connection and XML-RPC in the same thread would defeat the purpose of handling all flows in the same thread.

5.8 Traffic Generation

Based on the discussion in the Traffic generation in the section 4.3, the flowgrind has a simple and efficient traffic generation system. The flowgrind controller has the separate option for the traffic generation as show in the figure 5.8, to support all the options as discussed in the section 4.4. According to the mathematical model discussed in section 4.2, the flowgrind are provided with random seed, which initializes the random number generator. If no seed is passed to the flowgrind, then flowgrind uses random number provided by the OS. For each and every flow, the random number is reinitialized. The flowgrind relies on the LibGSL [27] for the calculation of values of the random number distribution. See the table 5.1 for the over all distribution possible in the flowgrind.

After initializing the random number, the daemons at sender side runs over the select() loop, and call the write_data(), if there is any file descriptor for write operation. The flowgrind daemon generates the block size for the response size and request size according to the distribution. If the inter departure time is set, then the daemon wait for that duration and then write the data into the socket.

The daemon in the receiver side also runs over the select () loop, and read the block, once the file descriptor is ready to read. Each data block have an application header as shown in the figure 5.4. If the requested_block_size is set, then receiver daemon writes the respond block. This mechanism is used to calculate the Application level RTT, which is discussed in the subsection 5.11.3. In this section, the discussion is limited to the fields this_block_size and requested_block_size.

Table 5.1: Probability distribution available in flowgrind.

Distribution	Parameters	Native support	LibGSL support
Constant	value a	✓	✓
Uniform	min a, max b	✓	✓
Exponential	mean μ	-	✓
Normal	mean μ, variance σ^2	-	✓
Log normal	mean μ, standard deviation σ	-	✓
Pareto	shape x_{min}, scale k	-	✓
Weibull	shape α, scale β	-	✓

this_block_size: This is size of the current block size. This provide the information regarding the boundaries between two consecutive blocks and help to find the application header for the each blocks.

requested_block_size: This represents the size of the response block requested by the sender to the receiver. If it is set to -1, then it indicates the sender, that it is response block from the receiver.

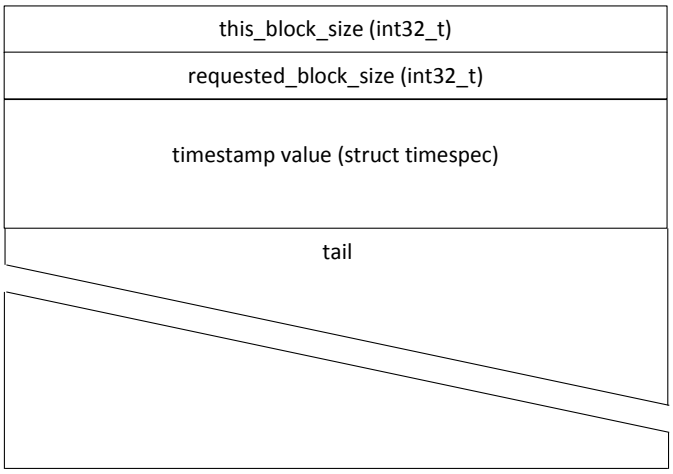

Figure 5.4: Flowgrind application header

5.9 Rate - limited flows

The Flowgrind provides the rate-limited flows features in them. This feature enables the flowgrind to generate the definite load. The rate limited feature emulates the live multimedia communication with the steady throughput and round trip value [62].

The rate limiting feature is done on the source host. The destination host processes the request as soon as it gets the data block from the source host. The flow is rate-limited by using the stochastic feature discussed in the section 4.4. The interdepature time is used to introduce the additional delay between the two data blocks, there by reduces the data transfer between the source and destination.

5.10 Work flow in the flowgrind

The workflow in the flowgrind can be divided into 3 major parts controller and data connection, flow operation in the daemon and test report display in controller.

5.10.1 Controller and Data connection

The flowgrind controller initializes the flow structure and parse the testing parameter from the command line argument by parsing general options and the flow options as shown in the figures 5.5 and 5.6. The flowgrind controller checks the flowgrind daemon

state and the flowgrind release version in the daemon and if there is a mismatch then corresponding warning sign, is issued in the controller terminal.

Before understanding the data test connection between the flowgrind daemons, one has to understand that the flowgrind daemon runs the daemon in two difference process. One process is used to communicate with the flowgrind controller through the XML-RPC with default port number 5999, and the next process is actually used to run the data test connection between the daemons. The user only has the control to bind the interface and the port only for XML-RPC. The flowgrind daemon data connection is not controlled by the user. The flowgrind by itself set the port connection for the test data connection between the daemons. By default the flowgrind daemon runs in the *wildcard address*, until it bind explicitly to an IP address.

The communication between the two process is done by pipe(), this is the unidirectional data channel between the flowgrind XML-RPC connection process and flowgrind data connection process. The pipe file descriptor is used to read at one end of the pipe and write in the other end of the pipe. Once the flowgrind XML-RPC process get the test parameter from the flowgrind controller, it will write the data in the pipe file descriptor and the flowgrind data test process read the data from the another end of the pipe. The data remains in the write end of the pipe by kernel buffering until it is read by the read end of the pipe. The separate thread is used to dispatch the request from XML-RPC connection process to the test data connection.

The flow preparation for the source daemon and the destination daemon is done separately by the controller. First the flowgrind controller queries the flowgrind daemon through the XML-RPC connection and pass the test configuration parameter to daemon through the add_flow_destination(), test configuration parameter includes destination host binding address, initial delay, test duration and reporting interval for report from the daemon to the controller.

Once the XML-RPC is established between the controller and the flowgrind daemon for the XML-RPC connection process, then the flowgrind daemon dispatch the request to the flowgrind daemon test connection process. The flowgrind daemon test connection process actually bind the controller with the given flowgrind interface address and the listen port set by the kernel in the socket.

The flowgrind daemon data connection process send back the listen port information back to the flowgrind control connection process through the pipe interprocess communication. Then the controller gets this information from the destination daemon and set this value as the destination address and destination port to the source flowgrind daemon. And the source flowgrind daemon connects to the destination address and destination port. Then the destination flowgrind daemon accepts the data test connection with source flowgrind daemon.

5.10.2 Test flow operation

The flowgrind controller starts the test flow in the both source and destination daemon endpoints, after that the data connection between the source and destination daemon is established. The flowgrind daemon sets the test duration and report interval timeout and start the test flow between the source and destination. The flowgrind daemon runs in a non-blocking select -loop over the socket file descriptor over all the active flows. Depending upon the read and write file descriptor, the corresponding function read_data() or write_data() is called. In addition to the read and write operation, select – loop also look for the pipe read end to get the data from the flowgrind daemon XML-RPC connection process. The flowgrind daemon is used to check the timer value in a loop, if the timer value exceeds the timer report interval, then the flowgrind daemon reports the interval flow report. When the test runs, the flowgrind daemon periodically sample and collect the report for the each flow. The flowgrind daemon make the report statistics to maintain the records for the interval and final report. The interval statistics records are reset after sending the report interval to the controller. But final report, aggregates all the report records.

The flowgrind controller fetches the reports for each flow from the daemon using the XML-RPC connection. Then the flowgrind controller decode the report and break it down to individual performance metrics, and then the performance metrics is used to display the report interval in the controller host. If either the flow gets completed successfully or exited due to the error, then the final report can be fetched from the daemon and displayed in the flowgrind controller host. One can also create the log file for the report using the controller option as shown in the figure 5.5. Once all the active flows are completed in the daemon, the daemon returns to the idle state. The daemon can again further run the test.

5.10.3 Read and write operation

The flowgrind maintains a separate application header for each data block in the test data connection. For more information refer the section 5.8 and the figure 5.4. So in the write operation depending upon the stochastic process, the block size is written in the data block. Before writing the data into the socket, the source daemon fills the 3th field in the application header with the timestamp values and then checks the interdepature gap as discussed in the section 5.9 to introduce the delay between the next block for the rate limitation.

In read operation, the flowgrind reads the block data with the minimum block size of 40 bytes to read the application header. This information is used to identify the data block as a request block or response block. If the block is the response block, then the round trip value is calculated and the result is aggregate in the report. And if the block is the request block then Inter Arrival Time is calculated.

5.10.4 Controller reporting procedure

Once the test data connection is established between the source and destination daemon as discussed in the subsection 5.10.1. The flowgrind controller fetch the reports from the daemon and suspends itself from the execution by calling usleep() for the report interval and then fetch the reports from the controllers. The controller fetches the reports from all the flow endpoints in the test. Each flow in the flowgrind could have different test duration and reporting time interval. This scenario is also taken into the consideration by the flowgrind controller and fetches the results according to their time duration, interval report and if the tests are set with the initial time delay. Once the flow test is completed, the flowgrind controller fetches the final reports from the each flow endpoints, and displays the final results at the end of the test. After this flowgrind controller tears down all XML-RPC connection with the daemons and terminate itself.

5.11 Performance metrics measurement

In this section, we will discuss regarding the performance metrics measurement for the throughput, and round-trip time in the flowgrind, netperf and Iperf.

5.11.1 Throughput

As discussed in the section 2.4, the standardization of the bulk transfer capacity (BTC) from the RFC 3148 [39], where the BTC is defined as follows.

$$\text{BTC} = \frac{\text{data\_sent}}{\text{elapsed\_time}}$$

According to this definition, the flowgrind calculated the elapsed time by taking the time interval between the daemon report begin timestamp and daemon report end timestamp. The data sent is calculated and aggregated after the each write operation. The throughput calculation methodology in the flowgrind, netperf and iperf are the same.

5.11.2 Round-trip time

Netperf can test the round trip time measurement for both TCP and UDP protocol using the TCP_RR and UDP_RR tests. The round trip time calculation is done at the application level. So the latency measurement in this test involves the entire layer below the application level. It includes kernel Network stack and the Network Interface Controller (NIC) device drivers. So the point should be noted out that the network stack contribute to the overall latency in the application layer. Netperf server and client

transfer only one byte of data between them in the request-response test. The actual throughput is defined as the number of transaction, the transaction is defined as the number of transaction between netperf client single request and netperf server single response.

$$\text{throughput} = \frac{\text{Number\_of\_Transacc}}{\text{elapsed\_time}}$$

Transaction rate is defined as the transaction per second and the Round trip time latency is calculated as follows

$$\text{Round Trip Time} = \frac{1}{\text{throughput}}$$

UDP Request-Response test(UDP_RR) works in the same way as the TCP_RR but the only difference between is that it uses connection-less sockets, but TCP_RR test includes connection-oriented socket. Iperf is used to measure the TCP throughput and jitter only in the UDP protocol. It doesn't support to measure the round trip time in the TCP.

5.11.3 RTT measurement in the flowgrind

RTT measurement in the flowgrind is done in the Application level for the request response test. As shown in the figure 5.4 from the section 5.8, each data block has an application header. This application header is used to develop the stochastic traffic model, but the interesting field for the RTT calculation in the application header are the 2nd field (requested_block_size), 3rd field to hold the timestamp values.The requested_block_size is used to indicate whether the current data block request responds by setting the field value as 0 or -1.

Timestamp field consists of a 16 byte timespec struct as the 3rd field to hold the timestamp value, which is grabbed just before writing the data block into the socket. This data block is transferred from the source daemon to the destination daemon through the socket connection. Once the data block is received at the destination daemon, the destination daemon parses the application header and looks for the request_block_size field. If the request_block_size is greater than the minimum block size, which is discussed in the section 5.8, then current block is identified as the response request block from the source daemon. Then destination daemon writes a new response block with request response size from the source destination. The new data block is set as the response block by setting the field as -1, then destination daemon copy exact timestamp from the request data block and then fill the same timestamp in response data block, which is sent from the destination daemon to the source daemon.

Once the response data block reach the source daemon, then the source daemon identify the current data block as the response block by parsing the 3rd field in the application

header. After the response block identification, source daemon gets the present timestamp from the host. The current timestamp and the timestamp value from the response block, give the round trip time for the current data block. The implementation of the round trip time is similar to the standard and methodology discussed in the RFC 2681, but the implementation part is done in the application level, which means that round trip also include the network stack over all latency and application level system call latency.

Table 5.2: Performance measurement tools feature matrix with flowgrind added

Feature	nuttcp	iperf	thrulay	netperf	flowgrind
TCP	✓	✓	✓	✓	✓
UDP	✓	✓	✓	✓	-
IPv6	✓	✓	✓	✓	-
RTT	✓	-	✓	-	✓
IAT	✓	-	✓	-	✓
Network Transactions/s	-	-	-	✓	✓
CPU utilization	✓	-	-	✓	-
third party tests	✓	-	-	-	✓
interval reports	✓	✓	✓	✓	✓
scheduling	-	-	-	-	✓
control/test interface separated	✓	-	-	-	✓
bi-directional traffic	✓	✓(pseudo)	-	-	✓
select congestion control	✓	✓	-	-	✓
TCP_INFO X(partial)	✓(partial)	-	-	-	✓
Rate Limiting	-	✓	UDP only	✓	✓
Request Response	-	-	-	✓(basic)	✓

5.12 Conclusion

In this chapter, flowgrind is introduced and discussed in details regarding its operation and architecture with traffic generation. From the table 5.2, which show the performance measurement matrix of the various tools with the flowgrind. From the table, the flowgrind has more advantages comparing to the other performance measurement tools. So the flowgrind is selected to implement the latency measurement module for this thesis project.

```
phobos2:~/flowgrind% ./flowgrind -h
Usage: flowgrind [OPTION]...
Advanced TCP traffic generator for Linux, FreeBSD, and Mac OS X.

Mandatory arguments to long options are mandatory for short options too.

General options:
  -h, --help[=WHAT]
                display help and exit. Optional WHAT can either be 'socket' for
                help on socket options or 'traffic' traffic generation help
  -v, --version  print version information and exit

Controller options:
  -c, --show-colon=TYPE[,TYPE]...
                display intermediated interval report column TYPE in output.
                Allowed values for TYPE are: 'interval', 'through', 'transac',
                'iat', 'kernel' (all show per default), and 'blocks', 'rtt',
                'delay' (optional)
  -e, --dump-prefix=PRE
                prepend prefix PRE to pcap dump filename (default: "flowgrind-")
  -i, --report-interval=#.#
                reporting interval, in seconds (default: 0.05s)
      --log-file[=FILE]
                write output to logfile FILE (default: flowgrind-'timestamp'.log)
  -m            report throughput in 2**20 bytes/s (default: 10**6 bit/s)
  -n, --flows=#  number of test flows (default: 1)
  -o            overwrite existing log files (default: don't)
  -p            don't print symbolic values (like INT_MAX) instead of numbers
  -q, --quiet    be quiet, do not log to screen (default: off)
  -s, --tcp-stack=TYPE
                don't determine unit of source TCP stacks automatically. Force
                unit to TYPE, where TYPE is 'segment' or 'byte'
  -w            write output to logfile (same as --log-file)
```

Figure 5.5: Flowgrind general options

```
Flow options:
  Some of these options take the flow endpoint as argument, denoted by 'x' in
  the option syntax. 'x' needs to be replaced with either 's' for the source
  endpoint, 'd' for the destination endpoint or 'b' for both endpoints. To
  specify different values for each endpoints, separate them by comma. For
  instance -W s=8192,d=4096 sets the advertised window to 8192 at the source
  and 4096 at the destination.

  -A x            use minimal response size needed for RTT calculation
                  (same as -G s=p,C,40)
  -B x=#          set requested sending buffer, in bytes
  -C x            stop flow if it is experiencing local congestion
  -D x=DSCP       DSCP value for TOS byte
  -E              enumerate bytes in payload instead of sending zeros
  -F #[,#]...     flow options following this option apply only to the given flow
                  IDs. Useful in combination with -n to set specific options
                  for certain flows. Numbering starts with 0, so -F 1 refers
                  to the second flow. With -1 all flow are refered
  -G x=(q|p|g):(C|U|E|N|L|P|W):#1:[#2]
                  activate stochastic traffic generation and set parameters
                  according to the used distribution. For additional information
                  see 'flowgrind --help=traffic'
  -H x=HOST[/CONTROL[:PORT]]
                  test from/to HOST. Optional argument is the address and port
                  for the CONTROL connection to the same host.
                  An endpoint that isn't specified is assumed to be localhost
  -J #            use random seed # (default: read /dev/urandom)
  -I              enable one-way delay calculation (no clock synchronization)
  -L              call connect() on test socket immediately before starting to
                  send data (late connect). If not specified the test connection
                  is established in the preparation phase before the test starts
  -M x            dump traffic using libpcap. flowgrindd must be run as root
  -N              shutdown() each socket direction after test flow
  -O x=OPT        set socket option OPT on test socket. For additional information
                  see 'flowgrind --help=socket'
  -P x            do not iterate through select() to continue sending in case
                  block size did not suffice to fill sending queue (pushy)
  -Q              summarize only, no intermediated interval reports are
                  computed (quiet)
  -R x=#.#(z|k|M|G)(b|B)
                  send at specified rate per second, where: z = 2**0, k = 2**10,
                  M = 2**20, G = 2**30, and b = bits/s (default), B = bytes/s
  -S x=#          set block (message) size, in bytes (same as -G s=q,C,#)
  -T x=#.#        set flow duration, in seconds (default: s=10,d=0)
  -U x=#          set application buffer size, in bytes (default: 8192)
                  truncates values if used with stochastic traffic generation
  -W x=#          set requested receiver buffer (advertised window), in bytes
  -Y x=#.#        set initial delay before the host starts to send, in seconds
```

Figure 5.6: Flowgrind flow options

```
phobos2:~/flowgrind% ./flowgrind --help=socket
flowgrind allows to set the following standard and non-standard socket options.

All socket options take the flow endpoint as argument, denoted by 'x' in the
option syntax. 'x' needs to be replaced with either 's' for the source endpoint,
'd' for the destination endpoint or 'b' for both endpoints. To specify different
values for each endpoints, separate them by comma. Moreover, it is possible to
repeatedly pass the same endpoint in order to specify multiple socket options

Standard socket options:
  -O x=TCP_CONGESTION=ALG
               set congestion control algorithm ALG on test socket
  -O x=TCP_CORK
               set TCP_CORK on test socket
  -O x=TCP_NODELAY
               disable nagle algorithm on test socket
  -O x=SO_DEBUG
               set SO_DEBUG on test socket
  -O x=IP_MTU_DISCOVER
               set IP_MTU_DISCOVER on test socket if not already enabled by
               system default
  -O x=ROUTE_RECORD
               set ROUTE_RECORD on test socket

Non-standard socket options:
  -O x=TCP_MTCP
               set TCP_MTCP (15) on test socket
  -O x=TCP_ELCN
               set TCP_ELCN (20) on test socket
  -O x=TCP_LCD set TCP_LCD (21) on test socket

Examples:
  -O s=TCP_CONGESTION=reno,d=SO_DEBUG
               sets Reno TCP as congestion control algorithm at the source and
               SO_DEBUG as socket option at the destinatio
  -O s=SO_DEBUG,s=TCP_CORK
               set SO_DEBUG and TCP_CORK as socket option at the source
```

Figure 5.7: Flowgrind socket options

```
phobos2:~/flowgrind% ./flowgrind --help=traffic
flowgrind supports stochastic traffic generation, which allows to conduct
besides normal bulk also advanced rate-limited and request-response data
transfers.

The stochastic traffic generation option '-G' takes the flow endpoint as
argument, denoted by 'x' in the option syntax. 'x' needs to be replaced with
either 's' for the source endpoint, 'd' for the destination endpoint or 'b' for
both endpoints. However, please note that bidirectional traffic generation can
lead to unexpected results. To specify different values for each endpoints,
separate them by comma.

Stochastic traffic generation:
  -G x=(q|p|g):(C|U|E|N|L|P|W):#1:[#2]
               Flow parameter:
                 q = request size (in bytes)
                 p = response size (in bytes)
                 g = request interpacket gap (in seconds)

               Distributions:
                 C = constant (#1: value, #2: not used)
                 U = uniform (#1: min, #2: max)
                 E = exponential (#1: lamba - lifetime, #2: not used)
                 N = normal (#1: mu - mean value, #2: sigma_square - variance)
                 L = lognormal (#1: zeta - mean, #2: sigma - std dev)
                 P = pareto (#1: k - shape, #2 x_min - scale)
                 W = weibull (#1: lambda - scale, #2: k - shape)
  -U x=#       specify a cap for the calculated values for request and response
               size (not needed for constant values or uniform distribution),
               values over this cap are recalculated

Examples:
  -G s=q:C:40
               use contant request size of 40 bytes
  -G s=p:N:2000:50
               use normal distributed response size with mean 2000 bytes and
               variance 50
  -G s=g:U:0.005:0.01
               use uniform distributed interpacket gap with minimum 0.005s and
               maximum 0.01s

Notes:
  - The man page contains more explained examples
  - Using bidirectional traffic generation can lead to unexpected results
  - Usage of -G in conjunction with -A, -R, -S is not recommended, as they
    overwrite each other. -A, -R and -S exist as shortcut only
```

Figure 5.8: Flowgrind traffic options

6 Implementation

This chapter discuss the implementation of the latency measurement in the flowgrind using the Timestamping feature in the linux. This chapter is divided into 2 sections, the section 6.1 discuss regarding the timestamp feature in linux kernel and Network Interface Controller (NIC) timestamping features in depth. The understanding of this section is essential for the latency measurement implementation in the flowgrind. The section 6.2 discuss regarding the constrain in the latency measurement implementation and also the solution for it.

6.1 Time stamping in Linux

Time stamping capability in the network interface card is used to keep the track of packet arrival and transmit it into the wire. The time stamping service in the network interface supports the affirmations of evidence that a datum existed before a particular system clock time [9]. NIC gets the incoming packets from the kernel, and these packets are time-stamped before they are sent out to the wire. Before discussing the NIC timestamping, it is essential to understand the Linux kernel timestamping control Interface [43].

6.1.1 Linux Kernel Timestamping control Interface

The kernel timestamping interface supports both the unidirectional (only receiving or transmitting timestamping) and bidirectional (supports both the receiving and transmitting) time stamping. The receiving network timestamps are listed as follows,

SO_TIMESTAMP: The timestamps for recording each arriving packet in system time. The timestamp are reported in the struct timeval (usec resolution), through the recvmsg system call in the form of ancillary messages(refer the subsection 6.1.6 for more information). This timestamps enables the socket option for time stamping the datagrams on the receiver side.

SO_TIMESTAMPS: The timestamping generation follows the same mechanism as the SO_TIMESTAMP, with one additional feature that the timestamp is reported as struct timespec (nsec resolution). This timestamping is used for the higher data rate NIC like 40 - 100 Gbit/s card. While using such NIC, it is recommended to use the struct timespec for the higher resolution (ns), than the struct timeval [45].

SO_TIMESTAMPING: It supports both the hardware and software time stamping for both transmission and reception. I also transmits and receives at the same time. This timestamping option supports the multiple request of the time stamp features at the same time. Since this timestamping option support multiple type, the input to the socket is given in the form of the bitmap flags, not as Boolean input parameter to the previous time stamping options [43].

6.1.2 Time stamping generation inside the kernel

The timestamp bitmap is requested to Linux kernel through the socket option to generate the timestamp from the network stack. All the combination of timestamping is valid in the Linux, but the NIC should support the combination of timestamping as well. Once the bitmap of flags are changed for the socket it is set to newly create packets, but this option is not applicable to the packets already existing in the network stack. This gives an advantage in the Linux kernel to do selective timestamping generation for the subset of packets by setting an send system call, between two socket set function to enable and disable the time stamping. For example, to do sampling with a subset of packets [43].

Timestamps feature is not only used for the reception and transmitting packet timestamping, but also used for another features as follows:

SOF_TIMESTAMPING_RX_HARDWARE: This feature enables the Linux kernel to get the NIC reception timestamps from the PTP Hardware clock in the NIC

SOF_TIMESTAMPING_RX_SOFTWARE: This option enables the Linux kernel to get the reception timestamps, when the packet enters the kernel. This means that timestamps are recorded just after the NIC driver hand over the packet to the kernel TCP/IP stack

SOF_TIMESTAMPING_TX_HARDWARE: This feature enables the NIC to get the transmit timestamps from the PTP Hardware clock in the NIC, just prior it leaves the device driver to the wire.

SOF_TIMESTAMPING_TX_SOFTWARE: This option enables the network interface card to get the reception timestamps, when the packet enters the NIC device driver. This means that timestamps are recorded just after the NIC driver gets the packet from the wire.

SOF_TIMESTAMPING_TX_SCHED: This enable the Linux kernel to get the transmit timestamp prior to entry point into the packet scheduler. The kernel packet processing time highly depends on the queueing delay in the kernel. The latency independent of protocol processing can be approximately found by using the timestamp difference between this timestamp and the software transmit timestamp. The overall latency in the kernel can be computed by taking time difference between this timestamp and the timestamp taken before and prior to the write or send function. In the host with the virtual devices, the packet used to transmit between

the different multiple devices, which indirectly means that packet travels through multiple packet scheduler. With this timestamp enabled, it generates timestamp at each layer, which can be used to the measure the fine grained of queueing delay [43].

SOF_TIMESTAMPING_TX_ACK: This timestamp generated when all the packets sent in the send buffer are acknowledged by the NIC. This timestamp supports only for stream socket, because it only supports the reliable protocol. In Linux kernel, this timestamp only supported for Transmission Control Protocol (TCP) currently. The timestamp is generated, when all the packets are acknowledged for the data sent from the send buffer. Thus, this support the cumulative acknowledge [43].

6.1.3 Reporting the timestamp value

The timestamp reporting feature in the Linux kernel is done by the 3 bits control, which controls the timestamps to be generated via the socket control message. These bit control is used to control the location from where the timestamps are to be reported from the stack. The timestamp are generated only to those packets, where the requests are made through the socket option in the generation request set. The Linux currently support the following timestamp options,

SOF_TIMESTAMPING_SOFTWARE: This timestamp is used to generate the timestamp support by the Linux kernel.

SOF_TIMESTAMPING_RAW_HARDWARE: This enable the timestamp at the NIC device driver level, the timestamping are generated by the PTP hardware clock inside the NIC card [14].

SOF_TIMESTAMPING_SYS_HARDWARE: Current Linux kernel ignored and deprecated this timestamp enable option, this is currently supported in the Linux 3.16. This is also used to generate the NIC timestamp and later convert it to system clock time. But today's NIC support the conversion inside the PTP hardware API, currently all timestamp supported NIC, do the conversion from the RAW PTP hardware clock timing to the system timing [38].

6.1.4 Additional options in the timestamping

Linux kernel provide additional options to support the timestamp feature than mere only generate timestamp values for the both transmit and receive packets. The additional feature options help to accurately identify the timestamp from the exact location from the kernel. In this subsection, we will discuss regarding the timestamping additional features,

SOF_TIMESTAMPING_OPT_ID: This feature generate an identifier along with each packet generation in the Linux, these identifier are unique to each and every packets. There would be multiple existing timestamp request for a process, in this case the packets could be reordered in the both receive and transmit path, while handling the packet in the network stack. For example, this could occur in the packet scheduler. So in this particular case the packet timestamp will also be queued out of order from the original queue in which it is send from the send or write function. Due to this problem, Linux get the timestamp of the reordered packet. So there must be mechanism, by which a process can identify which timestamp it belongs to and which packet is sent from the send buffer [43].

This feature associate each packet sent from the send buffer with an identifier, which is only unique for that particular send buffer packet. The unique identifier is generated by the u32 counter of socket. For the reliable protocol, like TCP, this unique identifier is based on the byte size of the send buffer. If the process transmit 8 byte of send buffer, then the unique identifier is 8. When the same socket send 2 byte of data into Linux kernel, then the unique identifier is 10. Unique identifier for the TCP timestamp is increment of byte size of the send buffer. For the datagram sockets, the identifier is just counter increment for each packet generate from the Linux kernel [43].

The socket counter starts, when this feature is enabled in the socket and the counter get reset, when the option is disabling in the socket. This feature supports sub sampling in the packets. The disabling option in the socket doesn't affect the unique identifier generated for the existing packets in the stack. This option only support the transmit timestamping packet in the Linux timestamping.

SOF_TIMESTAMPING_OPT_CMSG: It is a special case to coexist both the IP_PKTINFO information and timestamping information in the control message header. The control message is supported already for the all the timestamping feature. The process receives the timestamping information in the control message from the recv() or recvmsg().

SOF_TIMESTAMPING_OPT_TSONLY: This timestamp option only transmits the timestamp information in the control message with empty packet, which always piggybacked with all the control messages. This is supported only from the Linux kernel 3.19, this reduce the memory usage of the receive socket and this feature disable SOF_TIMESTAMPING_OPT_CMSG, so we won't get the IP_PKTINFO information simultaneously. In order to get good utilization of timestamp feature in the Linux, it is always recommended to use both the SOF_TIMESTAMPING_OPT_ID and SOF_TIMESTAMPING_OPT_TSONLY option [43].

But there is exception in the usage for these timestamp options; it is possible to use only SOF_TIMESTAMPING_OPT_TSONLY or SOF_TIMESTAMPING_OPT_CMSG, because SOL_IP/IP_PKINFO is used to detect

the outgoing packet information through the NIC, comes along with the original packet. So combining both options to co-exist is not possible [43].

6.1.5 Bytestream (TCP) timestamp in Linux

Linux timestamp option supports timestamping every byte in a bytestream, that is when a request is made to Linux to timestamp the bytestream. Then Linux recorded all bytes that have passed the timestamping point. For instance, SOF_TIMESTAMPING_TX_SOFTWARE takes timestamp of all bytes that leaves the Linux kernel to NIC device driver from the send buffer, despite how data buffer is converted into the packets.

Inside the Linux kernel, bytestream might be split bytes across the segments, and the segment can merge with each other, or reordered and bytes can exist simultaneously in multiple segment. So in general, bytestream has no boundary, so timestamping byte is not a trite issue. The timestamping option should be uniform in the implementation; otherwise comparing the timestamp result is not possible [43].

The implementation of timestamping and correlating it with the segment of byte result in consistent, if only both timestamping and timing measurement are chosen logically. Let us see the logical conclusion for the two scenarios. For instance,the acknowledgement timestamp is generated, when all bytes passed the timestamping point and other one is timestamping for IP Fragmentation in the device drivers. In case of fragmentation, only the first IP fragment is taken into the consideration for the transmit timestamp.

In case of TCP, it can simply break in 1:1 mapping, when travelling from the buffer to the skbuff due to the fact of existence of GSO, segmentation, Nagle, cork and auto cork in the transport control protocol. In this case, the individual last byte passed to send data buffer is tracked, even though it is not the last byte after the skbuff merge or extend operation. The skbuff stores only one sequence number in it structure tskey, so depending upon the tskey only one timestamp can be generated [43].

In the extremely unusual case, two requests for the timestamping are emerged into the same skb. In this case, by enabling the SOF_TIMESTAMPING_OPT_ID, we could compare the unique identifier and also the byte offset at the send time. The SOF_TIMESTAMPING_OPT_ID option ensures that the timestamp is generated only when the packet across the timestamping point.

6.1.6 Data Interpretation

In the Linux, the timestamp information is retrieved from the ancillary control message by using feature supported in the recv()/ recvmsg(), which sends back the control message data along with the read buffer. In this is section, we will look into the getting the

timestamp for the SOF_TIMESTAMP, SOF_TIMESTAMPNS and SOF_TIMESTAMPING option in the Linux.

Timestamp records

Timestamp records are stored in the SCM_TIMESTAMPING structure. The timestamp record is send back through the ancillary data feature in the recvmsg(), with the control message level, and control message type. The SCM_TIMESTAMPING structure support 3 timestamps in it. But the structure can hold only one timestamp at a time; it is a legacy feature in the Linux [43].

The ts [0] hold most of the timestamp record in it. The ts [1] used to store the hardware timestamp, which is converted to the system time. The direct hardware timestamp from the PTP hardware in the nsec resolution is passed in the ts [2]. The ts [2] can be directly used in the user space to get the raw PTP hardware clock timing information, which can be used to synchronize system time with the PTP hardware clock [43].

Transmitting timestamp to user space

The timestamp are transmitted to the user space by using the MSG_ERRQUEUE flag in the recv()/recvmsg(). The Linux timestamp module utilizes the socket's error queue information to send back the timestamp record from the Linux kernel network stack to the user space. And the control message buffer must be sufficiently large enough to hold the relevant metadata structure. The recvmsg function returns the two control message along with its incoming packet information from the Linux network stack [43]. As show in the figure 6.1 [29], the control message consist of timestamp information and also socket error information, which embedded into control message through the sock_extended_err structure. The user space distinguish the message using the message cm_level and cm_type, for the socket error information the cm_level is SOL_IP and cm_type is IP_RECVERR. For the timestamping information, cm_level is SOL_SOCKET and for the cm_type is SCM_TIMESTAMPING.

Timestamp type in the control message

The type of timestamping record is passed through the sock_extended_err ee_info. The SCM_TSTAMP define, from where the timestamp is generated inside the network stack. SCM_TSTAMP_SND is to pass the timestamp records to the skb from the PTP hardware clock timestamp. It supports both the SOF_TIMESTAMPING_TX_SOFTWARE and SOF_TIMESTAMPING_TX_HARDWARE timestamp.

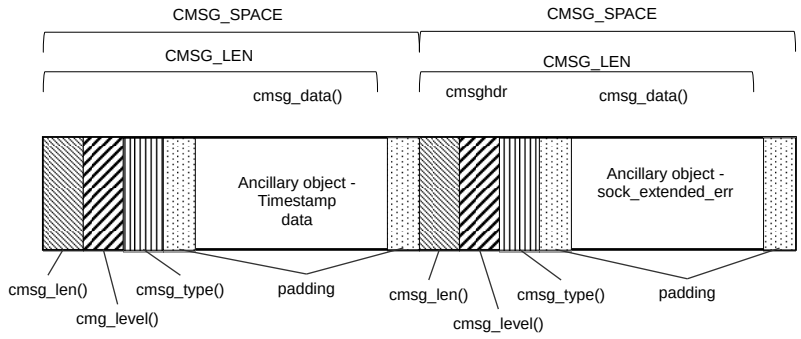

Figure 6.1: Ancillary data object in Linux timestamping

Timestamp in fragmentation

If the transmit packet is fragmented, then only first fragment is recorded for the timestamp and send back through the sending socket.

Reading the timestamp value

The message error queue is always read through the non-blocking operation. For the block waiting socket, poll or select function could be used. Poll has an advantage of using it return event - pollfd.revents. The poll() return the error events using the POLLERR, which mentions that the message error queue is ready to read.

Receiving timestamp

In the reception, there is no need to call the recvmsg() with the MSG_ERRQUEUE flag. Because the SCM_TIMESTAMPING is sent back to the user space through the ancillary control message along with receive buffer. This socket error doesn't contain the message for the SOL_IP/IP_RECVERROR [43].

6.1.7 Hardware Timestamp

The hardware timestamping support depends on the NIC device driver. The hardware timestamp could be checkout using ethtool tool with the option -T. The hardware timestamp requires admin privileged user to enable and disable time stamping by calling the ioctl function. Hardware time stamping is enabled by using the SIOCSHWTSTAMP to set and get the configuration information.

6.2 Latency measurement in Flowgrind

Hardware timestamping in the flowgrind is used to find precisely the network level latency in the networks. The development of the hardware timestamping is based on the usage of the kernel timestamping features support. For more information, regarding the timestamping feature in the Linux, refer the section 6.1. This section looks into the implementation of timestamp and methodology to calculate the Round Trip Time (RTT) measurement.

6.2.1 Enabling the Hardware timestamping

The hardware timestamping feature is enabled in the source flowgrind daemon, once the socket file descriptor is created with the flowgrind destination address and destination listen port. This file descriptor enables the hardware timestamping feature in the PTP hardware clock supported by the NIC. Refer to the section 6.1.2, to check the timestamping feature in the NIC.

Enabling the hardware timestamping feature with the Precision Time Protocol (PTP) hardware clock requires admin rights user to enable or disable the timestamp in an interface. The flowgrind daemon identifies the interface name in the host machine by using the flowgrind source daemon binds address. The flowgrind daemon get the interface name using the getifaddrs() system call to get the network interface name by parsing into the each element and comparing the address of interface with the flowgrind bind address to retrieve the interface name from the host machine. This interface name is used to make the interface request through the socket ioctl() to enable the hardware timestamp in interface using the SIOCSHWTSTAMP flag. Before making ioctl() call, As shown in the listing 6.1 flowgrind daemon configure the interface, to get the timestamp for outgoing packets and disable the incoming packet filter by using the following option. In this expression, tstconfig is the hwtstamp_config struct.

Listing 6.1: To enable Transmit timestamp for a net device

```
tstconfig.tx_type = HWTSTAMP_TX_ON;
tstconfig.rx_filter = HWTSTAMP_FILTER_NONE;
```

6.2.2 Enabling the time stamping feature in Linux

Time stamping feature in the Linux is not enabled by default, to enable the timestamping feature flowgrind call the setsockopt() with SO_TIMESTAMPING with timestamping multi flag bit. From the section 6.1.1, SO_TIMESTAMPING value is enabled as shown in the listing 6.2.

Listing 6.2: To enable the timestamps for Multi flags

```
rc = setsockopt(fd, SOL_SOCKET, SO_TIMESTAMPING, (void *) val, \&val);
```

val takes the multi flags to enable the timestamp value in the flowgrind. The flowgrind enables the timestamping feature in the Linux and also the timestamp value for the each acknowledgement with timestamp ID option enabled.

6.2.3 Timestamping procedure in the flowgrind

The sections 6.2.1and 6.2.2, discuss regarding the the hardware timestamping enabling for the NIC in Linux. This section discuss, how the timestamping works in the flowgrind, after enabling the option in the flowgrind. The section 6.1.2, which gives the overall picture of the timestamping support in the Linux. For the RTT measurement, flowgrind works mainly with the following timestamp options in the flowgrind.

- SOF_TIMESTAMPING_TX_ACK
- SOF_TIMESTAMPING_OPT_ID
- SOF_TIMESTAMPING_OPT_TSONLY

The SOF_TIMESTAMPING_TX_ACK is used to enable the acknowledgement for every data block transmitted from the NIC. For each data block acknowledgement, the PTP hardware clock timestamp value is stored in the scm_timestamping struct. Refer the section 6.1.6 and 6.1.6, for more information for the SCM_TIMESTAMP records. The timestamp for the acknowledgment supported in the Linux is applicable only for the reliable protocols - TCP.

The SOF_TIMESTAMPING_OPT_ID (refer the section 6.1.4) is used to give unique identifier for each data block transmitted from the network stack. Basically this feature is obtained from per socket namely u32 counter. For TCP protocol, this counter values increment every data block size. In the flowgrind, the default message size is 8192 byte, so the first data block ID has 8192 byte, the next message size also will be also 8192 byte, but second data block ID would be 16384.

The SOF_TIMESTAMPING_OPT_TSONLY is the additional extra feature provided in Linux 3.19 to get only the timestamp back from the message error queue (this is discussed in the section 6.1.6). This feature minimizes the amount of memory charged for the socket receive operation [43].

6.2.4 Processing the timestamp data

The timestamp values are processed from the network stack using the ancillary data structure in the Linux kernel. The application in the user space can retrieve the timestamp using the recv() system call with the MSG_ERRQUEUE flag.

Ancillary Data

The ancillary data basically used to send and receive the user credentials over the internet [29]. The ancillary data is the additional data along with the normal data. But the ancillary data concept is based on the certain defined format. This section discuss the details regarding the ancillary data usage in the flow grind.

Timestamp records in the Linux kernel

Timestamp value is piggybacked with the socket's error queue in the recvmsg() with flag MSG_ERRQUEUE flag. The overall picture regarding this ancillary data structure with timestamp option in Linux is showed in the figure 6.1. The recvmsg system call returns two ancillary data along with each socket error queue message. Each ancillary message has control message level (cmsg_level) to indicate the originating protocol and control message type (cmsg_type) to indicate the protocol specific type.

The first ancillary control message level consists of SOL_SOCKET and control message type SCM_TIMESTAMPING and consist of control message data with the struct scm_timestamping, this structure returns the timestamp value. The second ancillary control messsage level consist of SOL_IP or SOL_IPV6 and control message type IP_RECVERR or IPV6_RECVERR, and consist of control message data with struct sock_extended_err, this structure returns the Linux timestamp additional features like data block ID and timestamp type. As discussed in the section 6.1.6, where the SCM_TSTAMP_* (the * could be SCHED, SND, ACK) is used to identify the timestamp type. This information is stored in the struct sock_extended_err data member. The data block ID, which is returned by Linux via enabling the timestamp option SOF_TIMESTAMPING_OPT_ID is stored in the struct sock_extended_err data member. In addition to it, for more control message identification, the Linux kernel defines the timestamp socket message error queue with struct sock_extended_err data member ee_errno as ENOMSG and the data member ee_origin as SO_EE_ORIGIN_TIMESTAMPING.

Receiving the message error queue

The receiving process for the socket message error queue data from Linux kernel is done by using the recv() / recvfrom() /recvmsg() system call. The flowgrind uses the recvmsg(), because all other receive message system call doesn't have the capability to manipulate the ancillary data capability. So the recvmsg() is the natural component for receiving the ancillary data through the message error queue.

Listing 6.3: recvmsg function prototype

```
int recvmsg(int s, struct msghdr *msg, unsigned int flags);
```

From the listing 6.3, the msg is pointed to the message header structure, which includes the socket address members, I/O vectors references,and also ancillary data buffer members as show in the listing 6.4

Listing 6.4: Message header to hold control message

```
struct msghdr {
    void         *msg_name;
    socklen_t     msg_namelen;
    struct iovec *msg_iov;
    size_t        msg_iovlen;
    void         *msg_control;
    size_t        msg_controllen;
    int           msg_flags;
};
```

Where member msg_control points to the ancillary data buffer and msg_controller refers to the buffer size. Flowgrind timestamp module adjusts the ancillary data buffer, to hold sufficiently, the struct timestamp value and also the socket extended error information for getting the timestamp information regarding the timestamp type with unique ID information with each timestamp value.

But the recvmsg() processes only one socket error queue message at a time and it also slows down the overall transaction rate in the in the flowgrind. So flowgrind uses recvmmsg () instead of recvmsg() for receiving multiple messages, this benefits the performance in the flowgrind. In addition, it reduces the message overheads caused by the multiple call of the recvmsg().

Listing 6.5: recvmmsg function prototype

```
int recvmmsg(int sockfd, struct mmsghdr *msgvec, unsigned int vlen,
             unsigned int flags, struct timespec *timeout);
```

From the listing 6.5, the msgvec is the pointer to the array of struct mmsghdr and size of the array is defined in the vlen. The size of the array limits the number of message processed by the recvmmsg. In flowgrind, for 40GBit/s NIC, the maximum of 12 error queue messages are received by recvmmsg(), So the maximum limit to process the message error queue is set to 20. In additon to it, recvmmsg() has timeout parameter, but flowgrind follows non- blocking socket, so recvmmsg() reads as many as messages available at a single shot and returns immediately.

Pitfall in the pselect system call

In the initial development of flowgrind, the tool get hangs in write system call function in the flowgrind source daemon for the bulk transfer test and also get hang in recvmsg system call in the flowgrind source daemon for the request-response test. The reason for

this behavior is the pending message error queue in the socket. Though the pselect() says there is a file descriptor for the read operation, it turns out to be the message error queue. So calling recvmsg() to read the normal response data block cause recvmsg() to get hang. This the same for the write operation in the socket, which eventually fails because of the pending message error queue. This make flowgrind very difficult to debug. It appears that the select () couldn't distinguish both the read and write file descriptor on the same socket with the message error queue data from the socket.

This problem is also faced by the developer in the openswan -2.6.43 [19], released on 13th March 2015, it is an IPsec implementation and VPN software for Linux. In the openswan source code [19] in order to check the error message queue embedded in the ancillary data structure as control message for the message level SOL_IP and for type IP_RECVERR, the openswan application uses poll() in order to check the pending message queue error in the socket [19]. The poll() has the ability to distinguish read, write and error using the [event] field. The flowgrind uses poll() return event field to distinguish the events between the read, write and error. Flowgrind checks for the pending error message queue in the socket before the read and write operation. This feature give the flowgrind to process the message error queue messages efficiently.

6.2.5 Processing timestamp values

From the section 6.1.6, it was mentioned that flowgrind receives two ancillary data, with one having the timestamp records and another one having the timestamp type for each error message queue from the Linux kernel. The flowgrind processes the value if only both the ancillary data are present in the control message. The flowgrind uses the generic doubly linked list implementation to store the ancillary data according to the timestamp type.

For instance, the flow grind stores the timestamp value of data block transmitted from the NIC to the wire along with their unique ID in a separate linked list and the acknowledgement timestamp for the data block also with unique ID in a separate linked list. Since both the transmission and acknowledgement timestamp arrival from the Linux kernel to the user space is asynchronous, that is for example, the data block with unique ID 8192 is transmitted at the time TS1 and then NIC transmit the data block value with ID 16384 at TS2. But the acknowledgement for the unique ID 16384 would reaches the NIC before the unique ID 8192. So flow grind maintains separate data structure to store them.

6.2.6 Round trip time calculation

The above section 6.2.5 explains how the flowgrind handles and segregate the timestamp value for each type. This sub section discuss regarding the flowgrind calculation mechanism for the round trip time or two way delay.

From the RFC 2681,From the section 2.6, the two way delay is defined as the *"The two way delay dT between Src to Dst at time T, where dT is the time delay between Src sent to the first bit of packet, Dst at time T and Dst immediately sent, back to the packet to Src, and Src receive the last bit of the same packet at T+dT"*

The flowgrind adapts the same concept and methodology to measure the RTT based on the RFC 2681 definition. Flowgrind correlates with the *"time T"* as the timestamp values are taken, when the packet leave the Network interface card into the wire, and the *"time T+dT"* is taken with timestamp value, when the corresponding data block acknowledgement received in the NIC from the wire.

When the flowgrind gets the timestamp value from the Linux kernel, it will store the value in its data structure and calculate the RTT based on their unique ID, which is stored as the key and value pair in the linked list. For instance, if flowgrind receives the acknowledgement for the unique ID 24576, then it iterates through the transmit timestamp data structure to find the corresponding unique ID transmit timestamp value. The difference between the acknowledgement and transmitted timestamp value for the unique 24576 would give the RTT taken by the data block.

Depending upon the RTT value, both the minimum and maximum values are assigned and then the RTT is aggregated for the each transaction in the source flowgrind daemon. The number of transaction is calculated based on the number of acknowledgement received in the flowgrind daemon. So the average RTT is calculated by dividing the aggregated RTT by number of transactions done. The flowgrind stores the minimum, average and maximum RTT in the two data structure each for the interval report and final report. The interval report is reset after the each time the flowgrind daemon shared the interval report with the flowgrind controller.

Depending upon the timestamp option enabled, the flowgrind controller will shrink and expand, and display the results in the flowgrind controller. In the flowgrind controller, hardware enabled timestamp values are called as kernel level RTT, since the RTT is calculated based on the timestamp data from the Linux kernel.

6.3 Conclusion

According to the RFC 2681 [5], the flowgrind kernel level RTT value from the section 6.2.6 is similar to the definition for the *"wire time"* discussed in this RFC. The flowgrind Application RTT, which is discussed in the section 5.11.3, is similar to the definition of *"Host time"*. This is illustrated in the figure 6.2.The output of application level RTT and Kernel level RTT is shown in the figure 6.3. So this enables the environment to measure the difference between the Application level RTT and the kernel level RTT. The results are evaluated in the nextc chapter 7.

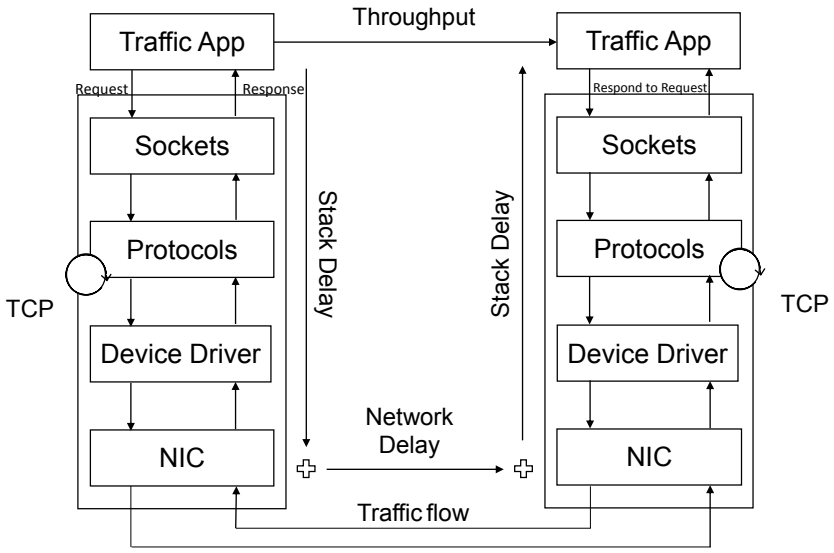

Figure 6.2: Flowgrind latency measurement

```
# Date: 2015-04-19-22:50:04, controlling host = phobos2.mgmt.muclab, number of flows = 1, reporting interval = 1.00s, [through] = 10**6 bit/second (flowgrind
      -0.7.5-141-gda2ff9)
# ID   begin     end      through   transac  min kRTT  avg kRTT  max kRTT  min aRTT  avg aRTT  max aRTT  uack  sack  lost  retr  tret  fack  reor  bkof  ca state  pmtu
#        [s]     [s]    [Mbit/s]    [#/s]      [ms]      [ms]      [ms]      [ms]      [ms]      [ms]   [#]   [#]   [#]   [#]   [#]   [#]   [#]   [#]                [B]
S   0  0.000   1.000  292.525531   565.98     0.040     0.136     0.224     0.400     2.259     2.774     0     0     0     0     0     0     3     0       open  1500
S   0  1.000   2.000  282.569247   543.00     0.025     0.129     0.211     1.904     2.335     2.906     0     0     0     0     0     0     3     0       open  1500
# ID   begin     end      through   transac  min kRTT  avg kRTT  max kRTT  min aRTT  avg aRTT  max aRTT  min IAT  avg IAT  max IAT  uack  sack  lost  retr  tret  fack
      reor  bkof  ca state  pmtu
#        [s]     [s]    [Mbit/s]    [#/s]      [ms]      [ms]      [ms]      [ms]      [ms]      [ms]     [ms]     [ms]     [ms]   [#]   [#]   [#]   [#]   [#]   [#]
         [#]   [#]              [B]
D   0  0.000   1.000   11.404581     0.00       inf       inf       inf       inf       inf       inf    0.002    0.028    0.319     1     0     0     0     0     0
         3     0       open  1500
D   0  1.000   2.000   11.041126     0.00       inf       inf       inf       inf       inf       inf    0.002    0.029    0.313     1     0     0     0     0     0
         3     0       open  1500
S   0  2.000   3.000  301.063710   566.01     0.041     0.132     0.492     1.167     2.547     4.395      inf      inf      inf     4     0     0     0     0     0
         3     0       open  1500
D   0  2.000   3.000   11.740216     0.00       inf       inf       inf       inf       inf       inf    0.002    0.027    0.475     1     0     0     0     0     0
         3     0       open  1500
S   0  3.000   4.000  285.712175   548.00     0.058     0.124     0.325     1.485     4.072     4.713      inf      inf      inf     4     0     0     0     0     0
         3     0       open  1500
D   0  3.000   4.000   11.157892     0.00       inf       inf       inf       inf       inf       inf    0.002    0.029    0.326     1     0     0     0     0     0
         3     0       open  1500
S   0  4.000   5.000  281.922543   540.96     0.059     0.122     0.232     3.565     4.158     4.698      inf      inf      inf     0     0     0     0     0     0
         3     0       open  1500
D   0  4.000   5.000   11.015083     0.00       inf       inf       inf       inf       inf       inf    0.002    0.029    0.323     0     0     0     0     0     0
         3     0       open  1500
S   0  5.000   6.000  284.368461   545.03     0.058     0.123     0.254     2.255     4.122     4.793      inf      inf      inf     4     0     0     0     0     0
         3     0       open  1500
D   0  5.000   6.000   11.105927     0.00       inf       inf       inf       inf       inf       inf    0.002    0.029    0.307     0     0     0     0     0     0
         3     0       open  1500
S   0  6.000   7.000  283.003321   542.99     0.061     0.125     0.267     2.891     4.140     4.792      inf      inf      inf     4     0     0     0     0     0
         3     0       open  1500
D   0  6.000   7.000   11.055017     0.00       inf       inf       inf       inf       inf       inf    0.002    0.029    0.320     0     0     0     0     0     0
         3     0       open  1500
S   0  7.000   8.000  283.167902   543.02     0.059     0.123     0.230     3.655     4.137     4.842      inf      inf      inf     0     0     0     0     0     0
         3     0       open  1500
D   0  7.000   8.000   11.059520     0.00       inf       inf       inf       inf       inf       inf    0.002    0.029    0.314     1     0     0     0     0     0
         3     0       open  1500
S   0  8.000   9.000  283.043341   542.96     0.058     0.124     0.243     3.626     4.140     4.791      inf      inf      inf     4     0     0     0     0     0
         3     0       open  1500
D   0  8.000   9.000   11.058662     0.00       inf       inf       inf       inf       inf       inf    0.002    0.029    0.321     0     0     0     0     0     0
         3     0       open  1500
S   0  9.000  10.000  284.445368   546.05     0.057     0.126     0.222     2.363     4.119     4.770      inf      inf      inf     4     0     0     0     0     0
         3     0       open  1500
D   0  9.000  10.000   11.109720     0.00       inf       inf       inf       inf       inf       inf    0.002    0.029    0.313     0     0     0     0     0     0
         3     0       open  1500

# ID    0 S: 172.18.120.21 (Linux 3.19.0.muclab+), random seed: 2408804610, sbuf = 16384/0 [B] (real/req), rbuf = 87380/0 [B] (real/req), SMSS = 1448 [B], PMTU =
      1500 [B], Interface MTU = 1500 (Ethernet/PPP) [B], duration = 10.000/10.000 [s] (real/req), through = 286.182089/11.174444 [Mbit/s] (out/in), transactions/s =
       34920.14 [#], request blocks = 349343/0 [#] (out/in), response blocks = 0/349201 [#] (out/in), Application level RTT = 0.400/3.593/4.842 [ms] (min/avg/max),
      kernel level RTT = 0.025/0.126/0.492 [ms] (min/avg/max), TCP_NODELAY
# ID    0 D: 172.18.120.22 (Linux 3.19.0.muclab+), random seed: 2408804610, sbuf = 16384/0 [B] (real/req), rbuf = 87380/0 [B] (real/req), SMSS = 1448 [B], PMTU =
      1500 [B], Interface MTU = 1500 (Ethernet/PPP) [B], through = 11.174782/286.075227 [Mbit/s] (out/in), request blocks = 0/349205 [#] (out/in), response blocks =
       349204/0 [#] (out/in), IAT = 0.002/0.029/0.475 [ms] (min/avg/max), delay = 0.000/1.558/2.682 [ms] (min/avg/max), TCP_NODELAY
```

Figure 6.3: Flowgrind measurement output: With Kernel level RTT and Application level RTT

7 Flowgrind Measurement Results

7.1 Methodology

The methodology for the implementation of the latency measurement and evaluating the network performance under heavy workload condition is based on the Request for Comment (RFC) 2681 from the section 2.6.1 and RFC 3148 from the section 2.4. The latency measurements in the flowgrind are discussed in detail in the section 6.2.6 and in the section 5.11.3. The measurements are done on real hardware device as described in the section 7.2, where tests are designed for different network conditions, with a set of realistic application based work load scenarios as listed in the section 4.4. The quantifiable and qualitative measurements are done using the flowgrind.

7.2 Testbest

The Thesis project tested is implemented by using the servers in the NetApp Munich High-Performance networking lab belongs to the Advanced Technology Group [40]. It uses to emulate the data center traffic, consist of six node cluster system of FAS3200 [57] and two-node FlashRay [56] prototype system. This infrastructure is used to emulate the data center traffic with the help of the traffic generation tool. The architecture for the the test bed is based on emulating the client-server and server-server architecture.

Testbed setup

This subsection discusses regarding the NetApp Munich lab, the Laurel is the network administrative server. It is a Fujitsu primergy RX100 S8 [28] with 1x4 cores with 32 GB RAM and 1.2TB local disk space over the FreeBSD-STABLE. It acts as the router and net boot server for the FAS 3270 [57] and Flash Rays. In our experiment, we used two-node Flash Ray prototype system, each consists of two SuperMicro servers [54] with 4x8 cores, and 128 GB RAM and network interfaces of 1 GB, 10GB and 40 GB which are used for our experiment. These nodes are named as Phobos 1 and Phobos 2. It also has the extension as Mora 1 and Mora 2. In our experiments, we used the Phobos and Mora to carried out with experiments. The cisco catalyst 2690S [12] with 48x 1G ports. This switch runs the commands and also used to control the network. It is placed in between the Laurel and other nodes in the lab. The nodes are connected to the 2690S for net booting and access over Secure Shell (SSH). The Flash Ray in the lab runs the

Debian GNU/Linux version 8(jessie). The OS is booted over the network through the laurel, which maintains Domain Name System (DNS) and Dynamic Host Configuration Protocol (DHCP). The normal storage box is called Cuba, where the home directory, backups and Debian images are maintained. The FlashRay nodes have a separate service processor (SP) that is used in the case of node failure through a serial console or through the SSH, for the instant to remotely cycle the power. The testing node FlashRay, Phobos and Mora is a purpose-built in all-flash storage architecture to explore and evaluate the high-performance, and the consistent in latency in a heavy workload condition. These nodes are supported with multiple Network Interface Controller (NIC) for the evaluating the network performance by varying the network traffic and load.

Measurement setup

The clock synchronization plays a vital role in the latency measurement between the nodes [15]. The Precision Time Protocol (PTP) protocol for clock synchronization is used in the measurement setup. To implement the PTP, the network stack has to support the hardware timestamp or software timestamp capabilities. In addition to it, NIC should also support hardware timestamp functionality in its physical hardware. The PTP can be implemented by the the Linux PTP [18] and PTP daemon (PTPd) [20], linuxptp is supported for the linux, but the PTPd supports both linux, FreeBSD and NetBSD. Since the Laurel is the network administrative server, where DHCP and DNS is maintained, PTPd was installed in the Laurel, which act as the master clock and rest of the PTP support equipments in the network, act as the PTP slaves and synchronizes to the master clock. All the Cisco switch and the Mellanox switch in the network supports the PTP hardware and Network interface nodes in Phobos and Mora, supports the PTP hardware clock and hardware timestamp, expect the management interface for the SSH. The PTP daemon was installed and started up automatically, in all the nodes and synchronized to the PTP master clock in the Laurel.

Phobos and Mora nodes are provided with the NIC from the vendors Intel, Mellanox. For the measurement setup of the project, the Intel card with 1Gbit/s and 10 Gbit/s capacity and Mellanox card is 40 Gbits capacity are used. The details of these interfaces are shown in the table 7.1.

The path of the measurement is between phobos1 $\leftarrow\rightarrow$ phobos2 and mora1 $\leftarrow\rightarrow$ mora2. This measurement setup is trying to emulate the server-client traffic and server-server traffic using the flowgrind application scenario as discussed in the section 4.4 and 5.8. The data center handles both the server-client traffic and sever-server traffic. So this measurement setup emulate the traffic to evaluate the performance of a data center with the help of a high performance FlashRay nodes.

Table 7.1: Ethernet Interface Overview.

Details/Data rate	1 Gbits/s	10 Gbits/s	40 Gbits/s
Product	I350 Gigabit Network Connection	82599ES 10-Gigabit SFI/SFP+ Network Connection	MT27500 Family [ConnectX-3]
Vendor	Intel Corporation	Intel Corporation	Mellanox Technologies
Capacity	1Gbit/s	10Gbits/s	40Gbit/s
Driver	igb	ixgbe	mlx4

7.3 Testing Scenarios

To evaluate the difference between the Application level Round Trip Time (RTT) and Kernel level RTT four different scenarios are created using the flowgrind traffic generation. The characteristics of the different applications are drew into a set of values for the traffic generation parameters. With the help of the traffic generation feature, as discussed in the section 4.4, the flowgrind generates a network workload, which emulate the application scenarios as referred in the section 5.8

Minimum Request-response test

The minimum request-response test is used to evaluate the application level RTT and kernel level RTT with the message size varying from 1KB to 1 MB. In this model, stochastic traffic generation distribution, where the request size is a constant distribution with message size varying from 1KB to 1MB and the response size with minimum block size of 40 bytes from destination to the source is shown in the table 7.2. The minimum block size consist of only the application header as discussed in the subsection 5.11.3, to calculate the application level RTT.

Table 7.2: Distributions and parameter values for minimum response scenario.

Parameter	Distribution (Values) [Maximum]
Request Size	Constant (1KB - 1MB)
Response Size	Constant (40B)
Interdeparture	Time Constant (0)
Socket Options	`TCP_NODELAY`

Request – Response: HTTP

As mentioned in the section 4.3, Hypertext Transfer Protocol (HTTP) is the widely used and the most prominent application layer protocol in the Internet [26]. Selecting the parameter for the traffic generation is based on the recommendation provided, for the HTTP traffic [1] and this represents the actual analysis of HTTP traffic experienced by the different ISPs. This model is already discussed in the section 4.4. The testing parameter are acculturated to match the flowgrind traffic generation model. From the table 7.3, the request size represent the `HTTP GET` request size and the response size distribution represents the size of the overall data delivered from the website server [1].

Table 7.3: Distributions and parameter values for the HTTP Scenario.

Parameter	Distribution (Values) [Maximum]
Request Size	Constant (350)
Response Size	Lognormal (μ = 9055, σ = 155) [100000]
Interdeparture Time	Constant (0)
Socket Options	`TCP_NODELAY`

Request – Response: SMTP

As mentioned in the RFC 2821 [37], Simple Mail Transfer Protocol (SMTP) is Simple Mail Transfer Protocol, which establishes the connection between Sender-SMTP and Receiver-SMTP and similar connection, established between the mail user agent and mail transfer agent. The typical scenario is that home user sends an email to the standard email server, and then email server response back, with the constant size. From the technical document [1], the parameter and distribution of the request size for the emulating SMTP represents the size of the email sent, approximately 100 email. The flowgrind SMTP scenario is shown in the table 7.4.

Table 7.4: Distributions and parameter values for the SMTP Scenario.

Parameter	Distribution (Values) [Maximum]
Request Size	Normal (μ = 8000, σ^2 = 1000) [60000]
Response Size	Constant (200)
Interdeparture Time	Constant (0)
Socket Options	`TCP_NODELAY`

Request – Response: Telnet

The flowgrind traffic emulates the telnet scenario as shown in the table 7.5, where the data is transferred from a shell session. For instance, typical data transfer in the Linux through the SSH and telnet. The parameter for the traffic generation is obtained by calculable analysis on the set of 20 telnet session. The value of the request and the response size within the distribution represents the size of data sent in the telnet session. The socket options with flowgrind option `TCP_NODELAY` disable the Nagle algorithm on test connection in this model. So all the data are sent out immediately, this is done to improve the response in the SMTP request-response model.

Table 7.5: Distributions and parameter values for the Telnet Scenario.

Parameter	Distribution (Values) [Maximum]
Request Size	Normal ($\mu = 2000$, $\sigma^2 = 500$) [20000]
Response Size	Normal ($\mu = 2000$, $\sigma^2 = 500$) [20000]
Interdeparture Time	Constant (0)
Socket Options	`TCP_NODELAY`

Request – Response : Rate – Limited: Streaming Media

The traffic generation modules in the flowgrind emulate the live streaming behavior. Based on the live dynamic streaming with Flash media server 3.5 [36], live streaming scenario are generated using total bit rate in Kbps for videos type D1 and HD, with 480p and 720p. For the live video streaming steady throughput and low RTT values are required for the smooth audio/live steaming. The parameter chosen are based on bit rate provided in the source, which usually occurs in Faster DSL and Cable modems [36]. A Normal distribution with emulate jitters in the data rate with constant request size, which introduce variable bitrate. The flowgrind live streaming scenario is shown in the table 7.6

Table 7.6: Distributions and parameter values for the Streaming Media Scenario.

Parameter	Distribution (Values) [Maximum]
Request Size	Constant (800)
Response Size	Constant (0)
Interdeparture Time	Normal ($\mu = 0.008$, $\sigma^2 = 0.001$)
Socket Options	`TCP_NODELAY`

7.4 Test schedule

Each application scenario is conducted for the 1 GBit/s, 10 GBit/s , and 40 GBit/s Network interface cards. For the minimum request-response test, a total of 18 test case scenarios generated for the 540 tests, request -response test for HTTP, SMTP, Telnet, and Streaming Media generates 180 test case in total 720 test cases are generated. The test runs for 20 iteration and test duration is for 30 secs. Each test is used to measure, both the application level RTT and the Kernel level RTT value. A test iteration run will be left out from the results evaluation, in case if any one of test fails. A test is considered failed if the required performance metrics is not found in the log.

All the test cases generate the log file. Test script is developed to run all the test case scenarios as discussed in the section 7.3, and another test script is developed to parse the log file for the performance parameter values and produces the results.

7.5 Results

7.5.1 Two way delay

The RTT, the time spent by a packets to travel forth and back in a network connection. For more information regarding the two way delay refer to the section 5.11.3 and section 6.2.6 for the implementation of RTT in application and in the kernel level. The difference between the minimum request-response test is the request message size is changed from 1KB to 1MB in the flowgrind is shown in the figures Nr 7.1, 7.2, 7.3 for the 1Gbit/s, 10Gbit/s and 40 Gbit/s NIC. The application oriented request-response tests - HTTP, TELNET, SMTP and Media streaming are based on the changes in the request and response size, and in the distribution. So the message data block size is varied to see the variation in the application level RTT and Kernel level RTT. Refer to the section 4.4 for more information regarding the traffic generation and from the tables 7.3 and 7.4, the HTTP makes large computed request-response size, when compared to the SMTP. The interquartile range of application RTT also increases along the dynamic request and response size(refer the figure Nr 7.4 and 7.7), because the Application level RTT includes the latency, which involves network stack and user space system call latency. The stack modules execution time also cause the overall latency in the measurement.

Application level latency includes the following contributors [21]

- Device drivers over the network adaptors
- Firmware in the network adaptor
- Operating system
- Network stack through which data reaches the application

- Test tool application through which data is calculated and converted into the performance metrics

Because of which the application level RTT measurement set, results is more deviated, and it's RTT value per iteration are skew or asymmetric distribution. The application level RTT skew distribution results in the tail away and ragged in one particular direction. The application level RTT values result in the potential and drastic change due to the skew distribution, where one set of values have a huge impact in the mean and the standard deviation. So all the results are shown in the median and interquartile range.

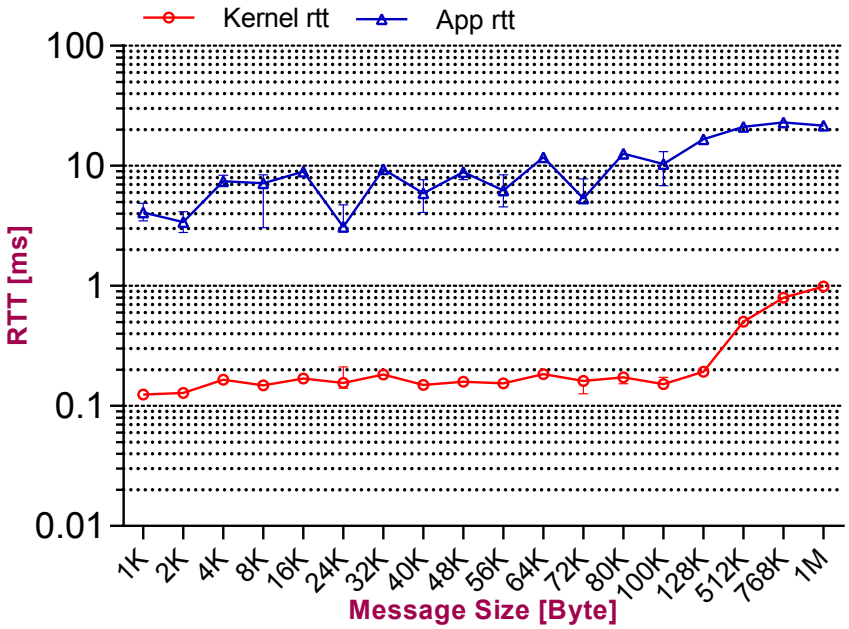

Figure 7.1: Minimum request response test with 1Gbits/s NIC

7.5.2 Performance result

Goodput is the most important performance metric for a network application. The linux timestamp enabling option in the flowgrind value affect prominently the number of transaction in the flowgrind and also the throughput value in the flowgrind. As discussed in the section 6.2.4, which explains the problem that the pselect function couldn't distinguish the message error queue, and normal read and write operation. Because of which poll function is introduced in addition to the pselect function to distinguish message queue error using the POLLERR. But the price to this solution came in the form of the performance issue in the flowgrind. For 1Gbit/s NIC, we getting only

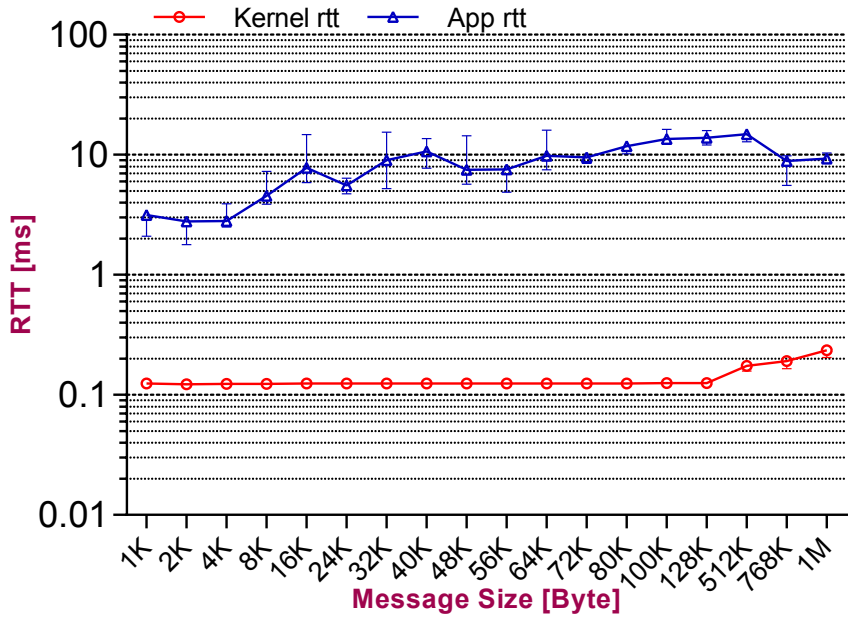

Figure 7.2: Minimum request response test with 10 Gbits/s NIC

an average of 180 Mbit/s, when comparing the same setup with the timestamp disabled option results in 900 Mbit/s for the message size of 16384 byte. This seriously raises the concern of the development of flowgrind timestamp feature for the RTT measurement.

The strace [22] system calls the trace tool for the flowgrind with and without timestamp enabled and the result shows that the pselect() has high system call latency of around 549 usec/call in the timestamp enabled flowgrind. In the second scenario, the pselect() has very lower system call latency of around 35 usec/call without timestamp feature enabled in the flowgrind. This obviously shows that the flowgrind spends most of the time in polling inside the pselect(). In order to further debug the issue, all pselect() and the associated Macro are replaced with the poll function. But the performance issue remains the same. The changing of the pselect function to poll function doesn't change the system call latency.

The flowgrind provides the traffic dumping feature as discussed in the section 5.6 using the option -M s, through which the users can analyze the dumped file. While using tcptrace [23] tool to analyze the dumped file shows that the interface spent an idle time of around 39 Milliseconds, which means that the time difference between the consecutive packets seen in the direction is relatively higher. This shows that flowgrind spends maximum system call latency time in poll or pselect function. This reduces the number of transaction and the throughput in the flowgrind.

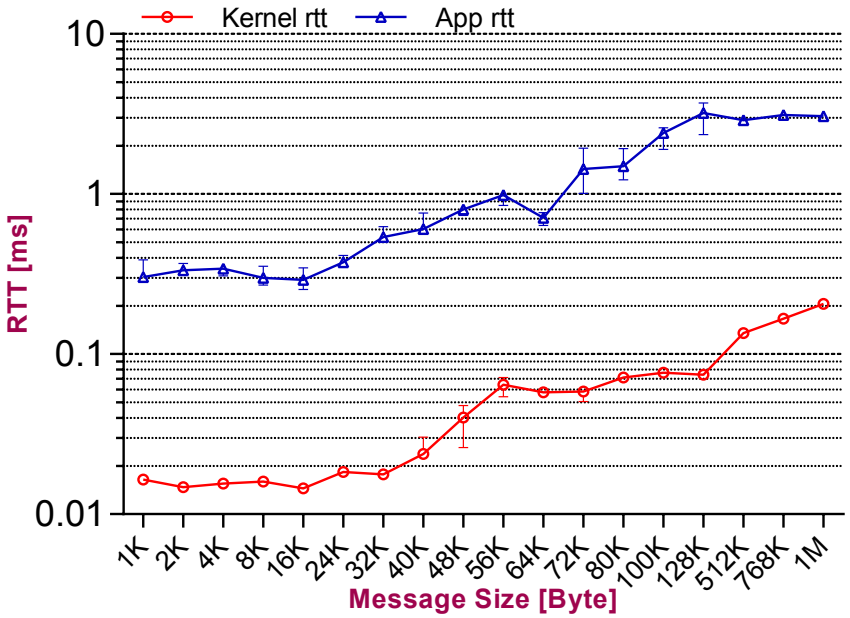

Figure 7.3: Minimum request response test with 40 Gbits/s NIC

Later the same set up is checked with the 100Mbit/s interface by limiting the speed of the 1GBbit/s Intel card and turning off the auto-negotiation using the ethtool [16] tool. The tcptrace analysis in this cause shows that the idle time is 1.3 ms and throughput is nearly 94 Mbit/s. This analysis shows that getting the timestamp from the NIC causes the throughput limitation in the flowgrind and this performance issue is discussed in the analysis section 7.5.3 and further discussion is carried out in the future work section 8.2

7.5.3 Analysis

The figure 7.3 for the minimum request response test shows that the application level RTT increases from 1KB to 1MB message size. This is because the application level RTT calculates the overall latency as mentioned in the section 5.11.3, which involves network stack, operating system and device driver, where the packets spends a significant amount of time on them. The TCP stack accepts the arbitrary messages that does segmentation and reassembly, and also might undergo fragmentation of segments at the router, to fit into the Maximum Transmission Unit (MTU).

In the case of long messages, the RTT undergoes segmentation and fragmentation, in addition to it, retransmission is also taken into the account. Hence application level RTT and kernel level RTT increase for the 1MB message size compared to the 1KB message

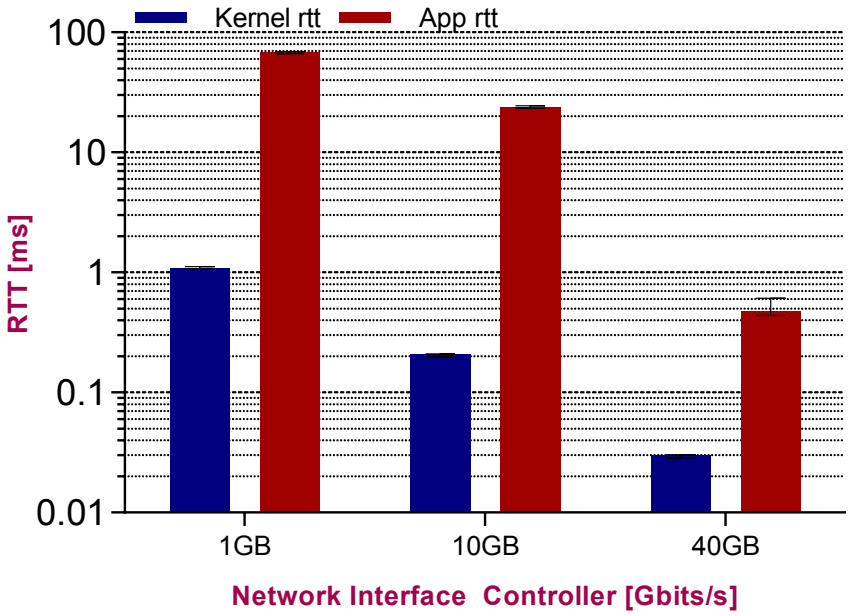

Figure 7.4: HTTP request response test

size. Also this evaluation is used to calculate the round trip at the NIC level, by using the PTP Hardware clock timestamp option. From the plot 7.2 for the Intel 10 Gbit/s NIC, the RTT from 1KB to 128KB message block exhibits less variation, but the RTT for the Mellanox Network Interface card, exhibits high variation than the Intel card. So this methodology is also used to evaluate the different NIC RTT.

The HTTP and Telnet plots 7.4and 7.6 show that the RTT values of both Application and Kernel level is get decreased from 1Gbit/s, 10 Gbit/s to 40 Gbit/s, where as the Media streaming plot 7.5 shows constant RTT, because request size is constant(800 byte) and interdepature time is a normal distribution.

One of the greatest challenges in this thesis was to find the root cause of the throughput limitation due to the activation of the timestamp in the Linux kernel. The behavior of the NIC with 1Gbit/s, 10Gbit/s and 40 Gbit/s are same, where the throughput is limited as shown in the plot 7.8. But in the 100Mbits NIC, the throughput is not limited. This is discussed in the previous subsection 7.5.2

Let us discuss the timestamping inside the Linux kernel and in the device driver, for instance in the Intel driver i40e, after cleaning the rx buffer, the device driver takes the timestamp value from the PTP hardware clock register values. In the Intel device driver, the i40e_ptp_tx_hwtstamp() is called to transmit the hardware timestamp value from the register and later this is converted into the kernel time from the raw PTP hardware

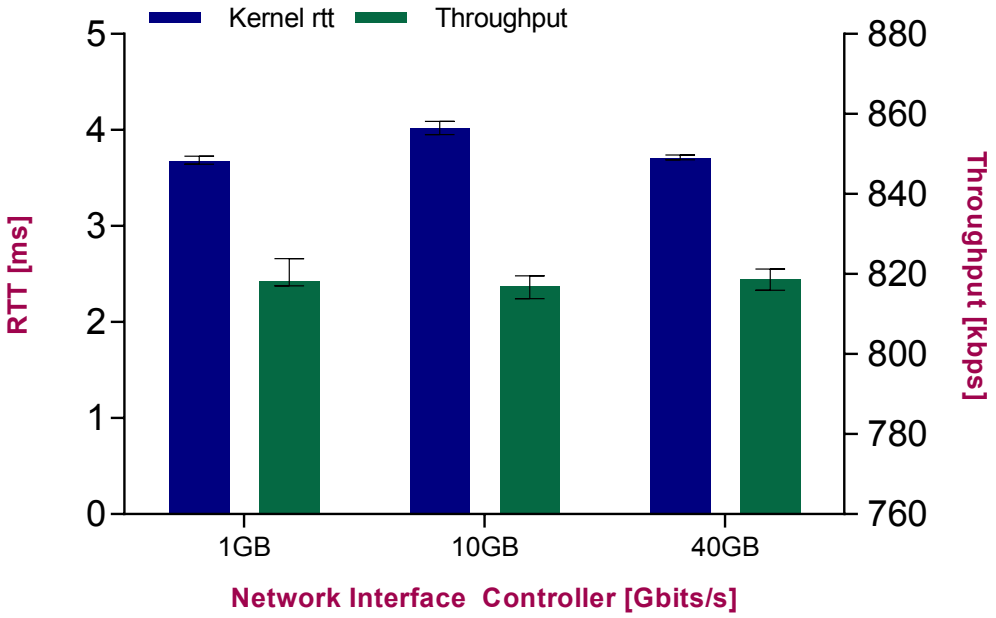

Figure 7.5: Streaming Media Rate-Limited test

timestamp, because the timestamp value is simply a 64 bit value which represents the nanosecond.

Later this value is filled in the struct skb_shared_hwtstamps. This structure is part of the struct skb_shared_info. The tcp_tx_timestamp() gets this information and share it in the user space by copying timestamp into the ancillary data. The process in getting the timestamp value from the PTP hardware clock is added as the overhead and cause the through put limitation in the flowgrind.

From the section 7.5.2, where the tcptrace output shows that the NIC spends around 39 ms as the idle time and also the pselect and poll system call latency also increased to 543 microsecs per call, when the timestamp is enabled in the flowgrind. Both the poll and the select system call handle the file descriptor in a linear manner. So the increase in the file descriptor causes them to go slower. In the timestamp enabled flowgrind, these system calls are waiting for the timestamping events that takes a definite time and causes a bottleneck

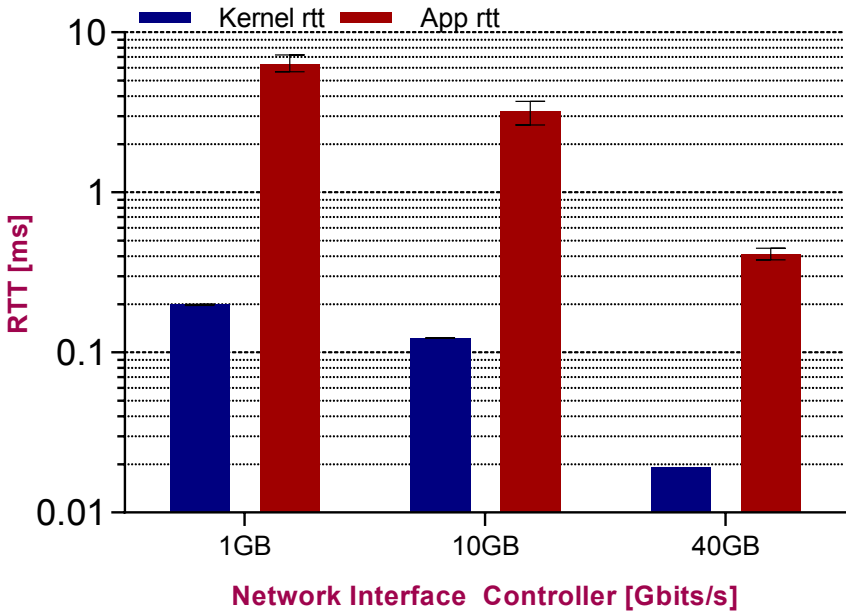

Figure 7.6: TELNET request response test

7.6 Conclusion

In this chapter a comprehensive evaluation of application level RTT and kernel level RTT is done using the flowgrind. It is explained in details with the minimum request-response and also by using the stochastic traffic generation to emulate the application scenarios for the HTTP, Telnet, SMTP and streaming media. The difference between the wire time and host time are discussed in the section 2.5.3 and here it is examined using the kernel level RTT and application level RTT. In addition to it, this chapter also discusses the performance issue due to the timestamp option enabled in the flowgrind.

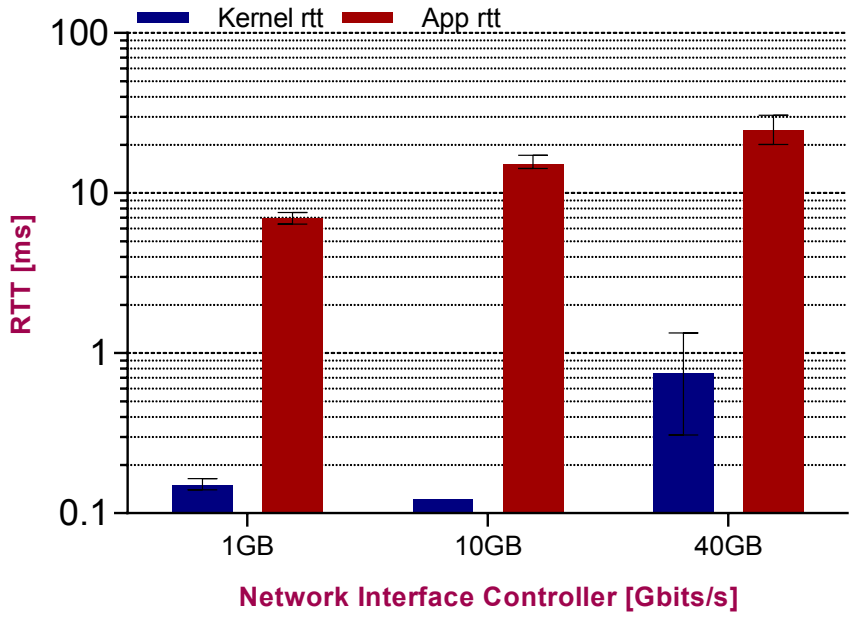

Figure 7.7: SMTP request response test

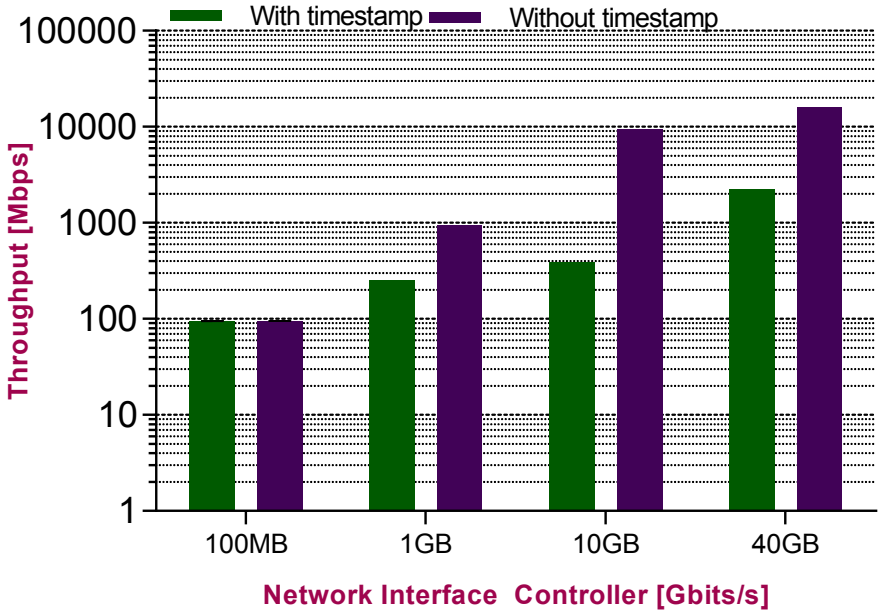

Figure 7.8: Goodput Measurement test

8 Conclusion

This chapter completes the thesis work in two parts: a summary of the objective of the thesis work and the achieved results, and some outstanding question to answer in the possible future work.

8.1 Summary

The initial objective of the Master thesis work is to find the latency measurement in the network, based on the standardization mentioned in the Internet Engineering Task Force (IETF) and to evaluate the latency on the basis of realistic and diverse internet traffic. The existing tools for the Round Trip Time (RTT) measurement are not quite standardized and didn't meet the requirements as mentioned in the standard working groups in the IETF. But flowgrind, the TCP/IP performance measurement tool extends the framework for the latency measurement and also provides the distributed architecture, unlike the other measurement tools, which provides only the client-server architecture. In addition flowgrind provides the benefit of using the stochastic traffic generation. The stochastic traffic generation is used to emulate the internet traffic, which other tools couldn't provide. So the latency measurement module was decided to be developed in the flowgrind.

The developed latency module in the flowgrind uses the timestamp feature in the Linux kernel to get the hardware timestamp values from the Network Interface Controller (NIC) and also to get the acknowledgement for the data sent from the NIC. The flowgrind utilizes the Precision Time Protocol (PTP) hardware clock present in the hardware timestamp enabled NIC, and to get the timestamp values through the device drivers. The developed latency measurement gives the RTT from the wire, which is similar to the term *"Wire Time"* as mentioned in the RFC 2679 and also discussed in details in the section 2.5.3. This extension in flowgrind gives the ability to evaluate the RTT measured in the application level by using the request-response framework, which is discussed in the section 5.11.3. The Application level RTT which is show in the flowgrind is similar to the term *"Host time"*, which is also discussed in the section 2.5.3. So evaluating both the application level RTT and kernel level RTT, gives the details regarding the latency handling in the firmware in network adapters, device drivers, OS, Network stack and Application in the user space.

In this thesis, we have achieved the measurement of the RTT from the Linux kernel and evaluate it along the with the application level RTT measurement in the different

scenarios, using the stochastic traffic generation feature in the flowgrind. The traffic generation emulated the Hypertext Transfer Protocol (HTTP), Telnet, Simple Mail Transfer Protocol (SMTP) and streaming media. In all these application scenarios, the RTT measurement is important for interactive application like remote shell sessions (discussed in the section 7.3), media streaming like cable modems (discussed in the section 7.3). The flowgrind demonstrates new technique to measure the latency, by following the procedures provided in the IETFs IP Performance Metrics (IPPM) working group and achieved the precise hardware level RTT measurement than the Application level RTT measurement.

8.2 Future work

There are concerns regarding the performance of flowgrind because of the timestamp feature enabled by it in the Linux kernel, which affect the throughput and number of transaction performance metrics measurement for the High speed NIC by increasing the system call latency in the polling function like poll() and select(). The timestamp feature enabled in the flowgrind cause the overhead, because of the reading hardware timestamp value from the PTP hardware clock register and copying the data into the sk_buff and then later to the ancillary data. This is the factor of concern for the high speed Network Interface card for example 1Gbit/s, 10Gbit/s and 40 Gbit/s. The flowgrind with the timestamp feature in the 100 Mbit/s as discussed in the section 7.5.2 shows that timestamp value causes the overhead only for the high speed NIC.

So there should be a light weight approach for the reading the PTP hardware clock timestamp values from the PTP registers, so that the timestamp records doesn't increases the idle time in the interface, that is the time gap between the two consecutive packets in the interface which is discussed in the section 7.5.2. By achieving this in future, it is possible to evaluate the goodput vs RTT, which is the most interesting evaluation to do. So far, there are no tools developed to measure the latency in the heavy work load condition. Tool like netperf also sends only 1 byte of data as the request-response model to get the RTT by using the number of transaction as discussed in the section 5.11.2. In addition to it, exploring the methods to access the file descriptor events through polling, which currently becomes a bottleneck in the high speed Network Interface Cards.

List of Abbreviations

BMWG	Benchmarking Methodology Working Group
BTC	Bulk Transport Capacity
CPU	Central Processing Unit
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DSL	Digital Subscriber Line
ECN	Explicit Congestion Notification
FTP	File Transfer Protocol
GB	Giga Byte
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
IEEE	Institute of Electrical and Electronic Engineers
IETF	Internet Engineering Task Force
IPPM	IP Performance Metrics
MSS	Maximum Segment Size
MTU	Maximum Transmission Unit
NIC	Network Interface Controller
NTP	Network Time Protocol
OS	Operating System
OSI	Open System Interconnection
OWAMP	One-way Active Measurement Protocol
PTP	Precision Time Protocol
RAM	Random Access Memory
RFC	Request for Comment

RPC	Remote Procedure Call
RTO	Retransmission Timeout
RTT	Round Trip Time
SCTP	Stream Control Transmission Protocol
SMTP	Simple Mail Transfer Protocol
SSH	Secure Shell
TCP	Transmission Control Protocol
TELNET	Teletype Network
TWAMP	Two-way Active Measurement Protocol
UDP	User Datagram Protocol
VoIP	Voice over Internet Protocol
WMN	Wireless Mesh Network
XML	Extensive Markup Language
XML-RPC	Extensible Markup Language Remote Procedure Call

Bibliography

[1] 3rd Generation Partnership Project 2 "3GPP2". *cdma 2000 Evaluation Methodology, Revision 0*. 2004. URL: http://www.3gpp2.org/Public_html/specs/C.R1002-0_v1.0_041221.pdf (visited on 12/10/2004).

[2] S. Van den Berghe A. Morton. *Framework for IP Performance Metrics*. RFC 5835. RFC Editor, Apr. 2010. URL: https://tools.ietf.org/html/rfc5835.

[3] Mohammad Alizadeh et al. 'Less is More: Trading a Little Bandwidth for Ultra-low Latency in the Data Center'. In: *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*. NSDI'12. San Jose, CA: USENIX Association, 2012, pp. 19–19. URL: http://dl.acm.org/citation.cfm?id=2228298.2228324.

[4] G. Almes, S. Kalidindi and M. Zekauskas. *A One-way Delay Metric for IPPM*. RFC 2679. RFC Editor, Sept. 1999. URL: https://tools.ietf.org/rfc/rfc2679.txt.

[5] G. Almes, S. Kalidindi and M. Zekauskas. *A Round-trip Delay Metric for IPPM*. RFC 2681. RFC Editor, Sept. 1999. URL: https://tools.ietf.org/html/rfc2681.

[6] H. Alvestrand. *A Mission Statement for the IETF*. RFC 3935. RFC Editor, Oct. 2004. URL: https://tools.ietf.org/html/rfc3935.

[7] IEEE 802.3 100 Gb/s Backplane and Copper Cable Study Group. *100 Gb/s Backplane and Copper Cable Task Force Documents*. URL: http://www.ieee802.org/3/bj/.

[8] S. Bradner. *Benchmarking Terminology for Network Interconnection Devices*. RFC 1242. RFC Editor, July 1991. URL: https://www.ietf.org/rfc/rfc1242.txt.

[9] D. Pinkas C. Adams P. Cain. *Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)*. RFC 3161. RFC Editor, Aug. 2001. URL: https://www.ietf.org/rfc/rfc3161.txt.

[10] J. Snell C. Grinstead. *Introduction to Probability*. URL: https://www.dartmouth.edu/~chance/teaching_aids/books_articles/probability_book/amsbook.mac.pdf.

[11] S. Cheshire. *White paper: Latency and the quest for interactivity*. Tech. rep. Volpe Welty Asset Management, L.L.C., Nov. 1996.

[12] Cisco. *Cisco Catalyst 2960 Series Switches*. URL: http://www.cisco.com/c/en/us/products/switches/catalyst-2960-series-switches/index.html.

[13] Cisco. *Using Test TCP (TTCP) to Test Throughput*. URL: http://www.cisco.com/c/en/us/support/docs/dial-access/asynchronous-connections/10340-ttcp.html.

[14] R. Cochran, C. Marinescu and C. Riesch. 'Synchronizing the Linux system time to a PTP hardware clock'. In: *Precision Clock Synchronization for Measurement Control and Communication (ISPCS), 2011 International IEEE Symposium on*. Sept. 2011, pp. 87–92. DOI: 10.1109/ISPCS.2011.6070158.

[15] Richard Cochran, Cristian Marinescu and Christian Riesch. 'Synchronizing the Linux system time to a PTP hardware clock'. In: *Precision Clock Synchronization for Measurement Control and Communication (ISPCS), 2011 International IEEE Symposium on*. IEEE. 2011, pp. 87–92.

[16] ethtool developers. *ethtool - utility for controlling network drivers and hardware*. URL: https://www.kernel.org/pub/software/network/ethtool/.

[17] libpcap developers. *lipcap*. URL: http://www.tcpdump.org/.

[18] Linux PTP developers. *The Linux PTP Project*. URL: http://linuxptp.sourceforge.net/.

[19] openswan developers. *openswan,IPsec implementation for Linux*. URL: https://www.openswan.org/.

[20] PTP daemon developers. *PTP daemon*. URL: http://ptpd.sourceforge.net/.

[21] Qlogic developers. *Introduction to Ethernet Latency, white paper*. 2014. URL: http://www.qlogic.com/Resources/Documents/TechnologyBriefs/Adapters/Tech_Brief_Introduction_to_Ethernet_Latency.pdf.

[22] strace developers. *strace system call tracer - debugging tool*. URL: http://sourceforge.net/projects/strace/.

[23] tcptrace developers. *tcptrace - tcp dump analysis tool*. URL: http://www.tcptrace.org/.

[24] D.Winer. *XML-RPC Specification*. 1999. URL: http://xmlrpc.scripting.com/spec.

[25] A. Morton E. Stephan L. Liang. *IP Performance Metrics (IPPM): Spatial and Multicast*. RFC 5644. RFC Editor, Oct. 2009. URL: https://tools.ietf.org/html/rfc5644.

[26] Rodger Edwards. 'Intelligent Buildings and Building Automation'. In: *Construction Management and Economics* 29.2 (2011), pp. 216–217. DOI: 10.1080/01446193.2010.542470. eprint: http://www.tandfonline.com/doi/pdf/10.1080/01446193.2010.542470. URL: http://www.tandfonline.com/doi/abs/10.1080/01446193.2010.542470.

[27] Free Software Foundation. *GSL - GNU Scientific Library*. URL: http://www.gnu.org/software/gsl/gsl.html.

[28] FUJITSU. *FUJITSU Server PRIMERGY RX100 S8 Mono socket 1U rack server*. 2014. URL: http://www.fujitsu.com/tw/Images/ds-py-rx100-s8.pdf.

[29] W. Gay. *Linux Socket Programming by Example*. By Example Series. Que, 2000. ISBN: 9780789722416. URL: http://books.google.de/books?id=xrhG0P91ZWkC.

[30] W3C Working Group. *Extensible Markup Language (XML)*. URL: http://www.w3.org/XML/.

[31] Seung-Sun Hong and S. Felix Wu. 'On Interactive Internet Traffic Replay'. In: *Proceedings of the 8th International Conference on Recent Advances in Intrusion Detection*. RAID'05. Seattle, WA: Springer-Verlag, 2006, pp. 247–264. ISBN: 3-540-31778-3, 978-3-540-31778-4. DOI: 10.1007/11663812_13. URL: http://dx.doi.org/10.1007/11663812_13.

[32] G. Huston. *Measuring IP Network Performance at The Internet Protocol Journal, Cisco Systems*. 2003. URL: http://www.cisco.com/web/about/ac123/ac147/archived_issues/ipj_6-1/ipj_6-1.pdf.

[33] Iperf. *Iperf performance performance tool*. URL: https://iperf.fr/.

[34] ixia. *IxNetwork™ IxCloudPerf QuickTest*. DATA SHEET 915190601. ixia, Oct. 2013. URL: http://www.ixiacom.com/sites/default/files/resources/datasheet/ixnetwork-ixcloudperf-quicktest.pdf.

[35] K. Yum K. Hedayat R. Krzanowski. *A Two-Way Active Measurement Protocol (TWAMP)*. RFC 5357. RFC Editor, Oct. 2008. URL: https://tools.ietf.org/rfc/rfc5357.txt.

[36] A. Kapoor. *Live dynamic streaming with Flash Media Server 3.5*. 2009. URL: http://www.adobe.com/devnet/adobe-media-server/articles/dynstream_live.html (visited on 09/12/2009).

[37] J. Klensin. *Simple Mail Transfer Protocol*. RFC 2821. RFC Editor, Apr. 2001. URL: https://tools.ietf.org/html/rfc2821.

[38] Zhi Li et al. 'A Hardware Time Stamping Method for PTP Messages Based on Linux system'. In: *TELKOMNIKA Indonesian Journal of Electrical Engineering* 11.9 (2013), pp. 5105–5111.

[39] M. Allman M. Mathis. *A Framework for Defining Empirical Bulk Transfer Capacity Metrics*. RFC 3148. RFC Editor, July 2001. URL: https://tools.ietf.org/rfc/rfc3148.txt.

[40] NetApp ATG Members. *Advanced Technology Group*. URL: https://atg.netapp.com/.

[41] Netperf. *Netperf performance performance tool*. URL: http://www.netperf.org/netperf/.

[42] NUTTCP. *NUTTCP performance measurement tool*. URL: http://nuttcp.net/nuttcp/beta/.

[43] P. Ohly. *Linux Document for timestamping (version 3.19)*. URL: https://www.kernel.org/doc/Documentation/networking/timestamping.txt.

[44] Diego Ongaro et al. 'Fast Crash Recovery in RAMCloud'. In: *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. SOSP '11. Cascais, Portugal: ACM, 2011, pp. 29–41. ISBN: 978-1-4503-0977-6. DOI: 10.1145/2043556.2043560. URL: http://doi.acm.org/10.1145/2043556.2043560.

[45] Peter Orosz, Tamas Skopko and Jozsef Imrek. 'Performance Evaluation of the Nanosecond Resolution Timestamping Feature of the Enhanced Libpcap'. In: *ICSNC 2011, The Sixth International Conference on Systems and Networks Communications*. 2011, pp. 220–225.

[46] John Ousterhout et al. 'The Case for RAMClouds: Scalable High-performance Storage Entirely in DRAM'. In: *SIGOPS Oper. Syst. Rev.* 43.4 (Jan. 2010), pp. 92–105. ISSN: 0163-5980. DOI: 10.1145/1713254.1713276. URL: http://doi.acm.org/10.1145/1713254.1713276.

[47] Zhao Rongcai and Zhang Shuo. 'Network traffic generation: A combination of stochastic and self-similar'. In: *Advanced Computer Control (ICACC), 2010 2nd International Conference on*. Vol. 2. Mar. 2010, pp. 171–175. DOI: 10.1109/ICACC.2010.5487204.

[48] J. McQuaid S. Bradner. *Benchmarking Terminology for Network Interconnection Devices*. RFC 2544. RFC Editor, Mar. 1999. URL: https://www.ietf.org/rfc/rfc2544.txt.

[49] A. Karp S. Shalunov B. Teitelbaum. *A One-way Active Measurement Protocol (OWAMP)*. RFC 4656. RFC Editor, Sept. 2006. URL: https://tools.ietf.org/rfc/rfc4656.txt.

[50] S. Shalunov. *thrulay-hd*. URL: http://thrulay-hd.sourceforge.net/.

[51] Stanislav Shalunov. *thrulay: Network Capacity and Delay Tester*. URL: http://www.internet2.edu/presentations/fall04/20040927-E2E-Shalunov.pdf.

[52] Stanislav Shalunov. *thrulay: Network Tester*. URL: http://www.internet2.edu/presentations/jt2006feb/20060207-thrulay-shalunov.pdf.

[53] M. Stiemerling. *IP Performance Metrics (ippm) charter-ietf-ippm-05*. URL: https://datatracker.ietf.org/wg/ippm/charter/.

[54] SuperMicro. *SuperServer*. URL: http://www.supermicro.com/products/system/.

[55] H. Frystyk T. Berners-Lee R. Fielding. *Hypertext Transfer Protocol – HTTP/1.0*. United States, 1996.

[56] NetApp Product team. *NetApp EF540 Flash Array*. URL: http://www.netapp.com/us/products/storage-systems/flash-ef540/.

[57] NetApp Product team. *NetApp FAS3200 Series*. URL: http://www.netapp.com/us/products/storage-systems/fas3200/index.aspx.

[58] Thrulay. *Thrulay performance performance tool*. URL: http://sourceforge.net/projects/thrulay/.

[59] C. Leres V. Jacobson and S. McCanne. *tcpdump*. URL: http://www.tcpdump.org/.

[60] J. Mahdavi V. Paxson G. Almes. *Framework for IP Performance Metrics*. RFC 2330. RFC Editor, May 1998. URL: https://tools.ietf.org/html/rfc2330.

[61] P.J. Winzer et al. '100-Gb/s DQPSK Transmission: From Laboratory Experiments to Field Trials'. In: *Lightwave Technology, Journal of* 26.20 (Oct. 2008), pp. 3388–3402. ISSN: 0733-8724. DOI: 10.1109/JLT.2008.925710.

[62] A. Zimmermann, A. Hannemann and T. Kosse. 'Flowgrind - A New Performance Measurement Tool'. In: *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*. Dec. 2010, pp. 1–6. DOI: 10.1109/GLOCOM.2010.5684167.

List of Figures

List of Tables