

Thesis no: XXXX-20XX-XX



# Performance analysis of TCP in a KVM virtualized environment

**Sasank Hyderkhan**

Faculty of Computing  
Blekinge Institute of Technology  
SE-371 79 Karlskrona Sweden

This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering. The thesis is equivalent to 20 weeks of full time studies.

**Contact Information:**

Author(s):

Sasank Hyderkhan,

E-mail: [sasank.sasank28@gmail.com](mailto:sasank.sasank28@gmail.com).

University advisor:

Dr.Patrik Arlos,

Department of Communication Systems,

Blekinge Institute of Technology, Sweden

Faculty of Computing  
Blekinge Institute of Technology  
SE-371 79 Karlskrona, Sweden

Internet : [www.bth.se](http://www.bth.se)  
Phone : +46 455 38 50 00  
Fax : +46 455 38 50 57

# ABSTRACT

The requirement of high quality services is increasing day by day. So, in order to meet up with this requirement new technologies are being developed one of them being virtualization. The main agenda of introducing virtualization is that though virtualization needs more powerful devices to run the hypervisor, the technique also helps to increase consolidation which makes efficient use of resources like increase in the CPU utilization. The virtualization technique helps us to run more VM's (Virtual Machine) on the same platform i.e. on the same hypervisor. In virtualization as number of VM's share the CPU will there be any effect on the performance of TCP with the performance influencing factors of virtualization. While TCP being the most widely used protocol and most reliable protocol can performance of TCP vary if different TCP congestion control mechanism are used in the virtualized environment are the main aims of this research.

In this study, we investigate the performance influencing factor of TCP in the virtualized environment and whether those influencing factors have any role to play with the performance of the TCP. Also which TCP congestion control mechanism is best suitable in order to download files when virtualization is used will be investigated by setting up a client-server test bed. The different TCP congestion control mechanism which have been used are CUBIC, BIC, Highspeed, Vegas, Veno, Yeah, Westwood, LP, Scalable, Reno, Hybla. Total download time has been compared in order to know which congestion control algorithm performs better in the virtualized environment.

The method that has been used to carry out the research is by experimentation. That is by changing the RAM sizes and CPU cores which are the performance influencing factors in virtualization and then analyzing the total download time while downloading a file by changing the TCP congestion control mechanisms by running a single guest VM. Apart from changing only congestion control mechanisms the other network parameters which effect the performance of the TCP such as Delay have been injected while downloading the file, to match up with the real time scenarios.

Results collected include average download time of a file by changing the different memory sizes and different CPU cores. Average Download time for different TCP congestion controls mechanisms with inclusion of the parameter that effects the total download time such as Delay.

From the results we got we can see that there is a slight influence on the performance of TCP by the performance influencing factors memory sizes and CPU cores allotted to the VM in the KVM virtualized environment and of all the TCP congestion control algorithms having TCP – BIC and TCP-YEAH performs the best in the KVM virtualized environment. The performance of TCP – LP is the least in the KVM virtualized environment.

**Keywords:** Hypervisor, Performance, Virtualization .

## **ACKNOWLEDGMENTS**

My deepest gratitude's and appreciation I would like to express to those who have encouraged me and contributed to the realization of this work. I am indebted to my advisor Patrik Arlos and his valuable ideas, which helped in the completion of this work. In fact he has guided on every crucial and important stages of this research work.

To my family members and my fellow MSc students and colleagues who one way or the other shared their support. Thank you.

Above all I would like to thank the Supreme Personality of God.

## LIST OF FIGURES

Figure 1.1: Types of Hypervisors .....	9
Figure 2.1: Performance influencing factors in virtualization .....	13
Figure 3.1: Experimental Setup .....	17
Figure 4: Download time for TCP-CUBIC on server side by varying delays .....	45
Figure 5: Download time for TCP-BIC on server side by varying delays.....	45
Figure 6: Download time for TCP-HIGHSPEED on server side by varying delays .....	45
Figure 7: Download time for TCP-VENO on server side by varying delays .....	46
Figure 8: Download time for TCP-VEGAS on server side by varying delays .....	46
Figure 9: Download time for TCP-YEAH on server side by varying delays .....	46
Figure 10: Download time for TCP-WESTWOOD on server side by varying delays .....	47
Figure 11: Download time for TCP-LP on server side by varying delays.....	47
Figure 12: Download time for TCP-SCALABLE on server side by varying delays.....	47
Figure 13: Download time for TCP-RENO on server side by varying delays .....	48
Figure 14: Download time for TCP-HYBLA on server side by varying delays .....	48

## LIST OF TABLES

Table 3.1: Device Specifications .....	19
Table 4.1: Comparison of download times for different CPUs and RAMs.....	20
Table 5.1.2: Comparison of download times by varying different CAs .....	23
Table 7.1: Comparison of download times by varying CPUs and RAM for 5ms delay.....	31
Table 7.2: Comparison of download times by varying CPUs and RAM for 10ms delay.....	31
Table 7.3: Comparison of download times by varying CPUs and RAM for 20ms delay.....	32
Table 7.4: Statistical analysis of different CPU and RAM values for delay 5ms.....	32
Table 7.5: Statistical analysis of different CPU and RAM values for delay 10ms.....	32
Table 7.6: Statistical analysis of different CPU and RAM values for delay 20ms.....	32
Table 7.7: Comparison of download time for different CAs the delay 5ms.....	33
Table 7.8: Comparison of download time for different CAs the delay 10ms.....	33
Table 7.9: Comparison of download time for different CAs the delay 20ms.....	34
Table 7.10: Values for CUBIC vs other TCP congestion algorithms.....	35
Table 7.11: Values for BIC vs other TCP congestion algorithms .....	36
Table 7.12: Values for VENO vs other TCP congestion algorithms .....	37
Table 7.13: Values for VEGAS vs other TCP congestion algorithms.....	38
Table 7.14: Values for YEAH vs other TCP congestion algorithms .....	39
Table 7.15: Values for WESTWOOD vs other TCP congestion algorithms.....	40
Table 7.16: Values for LP vs other TCP congestion algorithms .....	41
Table 7.17: Values for SCALABLE vs other TCP congestion algorithms .....	42
Table 7.18: Values for RENO vs other TCP congestion algorithms .....	43
Table 7.19: HYBLA vs other TCP congestion algorithms .....	44

## **ABBREVIATIONS**

<b>CA</b>	Congestion Algorithms
<b>CCA</b>	Congestion Control Algorithms
<b>COV</b>	Coefficient of Variance
<b>CPU</b>	Central Processing Unit
<b>FSM</b>	Finite State Machine
<b>KVM</b>	Kernel-based Virtual Machine
<b>LSVM</b>	Latency Sensitive Virtual Machines
<b>OS</b>	Operating system
<b>RAM</b>	Random Access Memory
<b>RTT</b>	Round Trip Time
<b>TCP</b>	Transmission Control Protocol
<b>VM</b>	Virtual Machine

# CONTENTS

<b>ABSTRACT</b> .....	<b>3</b>
<b>ACKNOWLEDGMENTS</b> .....	<b>4</b>
<b>LIST OF FIGURES</b> .....	<b>5</b>
<b>LIST OF TABLES</b> .....	<b>6</b>
<b>ABBREVIATIONS</b> .....	<b>7</b>
<b>CONTENTS</b> .....	<b>8</b>
<b>1 INTRODUCTION</b> .....	<b>9</b>
1.1 VIRTUALIZATION .....	9
1.2 KVM .....	10
1.3 TCP .....	10
1.4 RESEARCH QUESTIONS .....	11
1.5 RESEARCH METHOD .....	11
1.6 THESIS OUTLINE.....	11
<b>2 RELATED WORK</b> .....	<b>13</b>
<b>3 METHODOLOGY</b> .....	<b>16</b>
3.1 EXPERIMENTAL METHODOLOGY .....	16
3.2 EXPERIMENTAL PROCEDURE.....	16
3.3 HARDWARE AND SOFTWARE SPECIFICATIONS .....	19
3.4 EXPERIMENTAL SCENARIOS.....	19
3.4.1 Scenario A.....	19
3.4.2 Scenario B.....	19
<b>4 RESULTS AND DISCUSSION</b> .....	<b>20</b>
4.1 RESULTS FROM SCENARIO A.....	20
4.2 RESULTS FROM SCENARIO B.....	20
<b>5 ANALYSIS AND DISCUSSION</b> .....	<b>22</b>
<b>6 CONCLUSION AND FUTURE WORK</b> .....	<b>24</b>
<b>REFERENCES</b> .....	<b>25</b>
<b>7 APPENDIX</b> .....	<b>27</b>
7.1 EXPERIMENTS IN NTAS .....	27
7.2 PERL SCRIPT USED IN THE EXPERIMENT.....	27
7.3 NUMERICAL VALUES OBTAINED FOR SCENARIO A .....	31
7.4 NUMERICAL VALUES OBTAINED FOR SCENARIO B .....	33
7.5 GRAPHS FOR SCENARIO B .....	44

# 1 INTRODUCTION

TCP is a reliable protocol and virtualization is a technique whose usage is increasing day by day, with increase in the number of VM's the latency rates do rise as all the VM's share the common resources (CPU, Memory, Network, etc.) . This study is done to find out whether the allocated RAM or CPU cores affect the TCP performance in a virtualized environment. Furthermore to find out whether there is a TCP congestion control algorithm that performs better than the others in the KVM virtualized environment.

## 1.1 Virtualization

The requirement of high quality services is increasing day by day. So, in order to meet up with this requirement new technologies are being developed one of them being virtualization. Virtualization is a proven software technology that is rapidly transforming the IT field, cloud computing. The virtualization technique helps us to run more VM's (Virtual Machine) on the same platform i.e. on the hypervisor. The main agenda of introducing virtualization is that though virtualization needs more powerful devices to run the hypervisor as a number of VMs will be launched on the hypervisor, the technique also helps to increase consolidation which makes efficient use of resources like increase in the CPU utilization.

There are two type of hypervisors namely type-1 hypervisors and type-2 hypervisors. Type-1 hypervisors are the hypervisors which run directly on the hardware by controlling the hardware and managing the guest operating systems. Type-2 hypervisors are the hypervisors which run on an operating system. Examples of type-1 hypervisors are XEN and KVM and examples of type-2 hypervisors being VMware and virtual box[1].

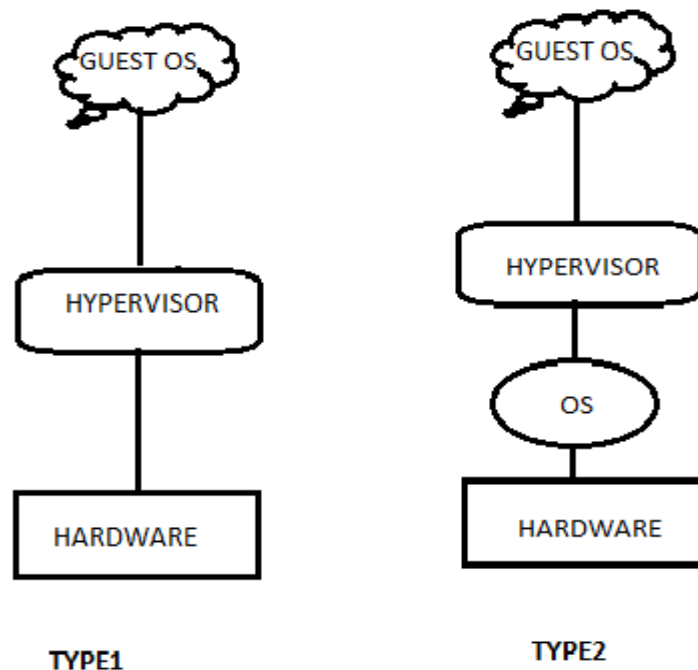


Figure 1.1: Types of Hypervisors

There are a number of virtualization techniques that are being used such as full virtualization, para virtualization, hardware-assisted virtualization and OS-layer virtualization.

**Full virtualization:** In full virtualization the guest operating system doesn't know that it's being virtualized and needs no modification. The hypervisor transforms all operating system instructions and stores the results for future use, while user level instructions run as they are without any change. Full virtualization offers isolation and security for virtual machines, and simplifies migration and portability[2].

**Para virtualization:** Para virtualization involves changing the operating system kernel to replace non virtualizable instructions with hypercalls that communicate directly with the hypervisor. The hypervisor also provide hypercall interfaces for other kernel operations such as memory management and time keeping[3].

**Hardware assisted virtualization:** The guest operating system doesn't know that it's being virtualized. In this virtualization technique the privileged and sensitive calls are set to automatically trap to the hypervisor, removing the need for either binary translation or para-virtualization[3].

## 1.2 KVM

KVM, Kernel-based virtual machine is a full virtualization solution for linux containing virtualization extensions and an open source software. In KVM the linux kernel itself acts as a hypervisor. All the capabilities of what a hypervisor can do are integrated into a host linux kernel, because of which it can simplify management and improve the performance in virtualized environments[4].

This approach of introducing the capabilities of the hypervisor into a linux kernel itself has a number of advantages such as all the ongoing work can be handled from the linux kernel itself in the virtualized environment. Apart from user mode and kernel mode which are traditionally used in a linux process KVM has another mode which is called as the guest mode. Guest mode has its own kernel and user-space variations[4].

KVM does come in different flavors such as full virtualization and para virtualization. By using 'virtio' a virtualization standard the guest operating systems i.e. the virtual machine can get most of the performance benefits of para-virtualization [5]. In the present research para virtualization is used as the virtualization technique. The reason being para virtualization performs better than the full virtualization because by using para virtualization the hardware devices that are required are only emulated unlike in full virtualization. Also to connect VMs to the physical network on the host layer and to share access to physical network cards on the physical machine a way is needed. The default one is to use a bridge device. Every instance running on the same physical machine goes through this bridge device in order to access the network, as well as the hypervisor host operating system. Apart from bridging there are different way to connect to the network while using the KVM hypervisor such as Open vSwitch, macvlan.

## 1.3 TCP

TCP, Transmission Control Protocol the transport layer protocol is a most widely used protocol for data services like file transfer, e-mail and WWW browser[6]. Most of the applications around the world use this protocol for communication. The main reason why this protocol is widely used because of its simplicity and it ensures reliable delivery of the data packets from one node to another through a series of electronic handshakes and acknowledgments, both nodes check data integrity and produce a "receipt of delivery".

The performance of TCP is important as it is an adaptive protocol to operate the network at the point of greatest efficiency according to the available network bandwidth. So better the

TCP performance better the quality of the network otherwise poor TCP performance may lead to very poor throughput, high latency rates. A number of TCP congestion algorithms have been developed such as CUBIC, BIC ,HIGHSPEED, VENO, VEGAS, YEAH, WESTWOOD, LP, SCALABLE, RENO, HYBLA. These congestion control algorithms address one or more network related issues such as high loss, long delays etc.

## 1.4 Research Questions

Aim of this research is to answer the below research questions. The research questions are answered by the experiments performed.

- a. Does the performance of TCP get affected by the allocated RAM or CPU cores in a virtualized environment?
- b. Which congestion control algorithm of TCP behaves ‘better’ in a virtualized environment?

## 1.5 Research Method

In order to do this research an experimental physical model has been taken up and the reason for choosing this sort of approach is clearly mentioned in the methodology section.

In order to answer the research question (a) that is to find out what affects the TCP performance of a VM, the allocated memory or CPU, we will execute some experiments. In the experiments we change the memory sizes and the CPU cores in the KVM virtualized server and download a file and see whether there are any changes proportional to changes in RAM sizes and CPU cores. We see to it that there is only one VM running on the hypervisor so that there aren’t any competition for resources. While conducting these experiments the congestion control algorithms in the client and server are set to the default congestion control algorithm that is CUBIC.

To find out which CCA behaves best in the virtualized environment we will change the congestion control algorithms both on the server side and the client side. Depending on the total download time taken to download the file we decide which congestion control algorithm will be performing the best among the different congestion control algorithms chosen. Some of the CCA’s have been developed in order to address the network issues such as long delays. In order to provide fairness i.e. equality to all the CCA’s a shaper is also introduced in order to provide delay within the network. The RAM sizes and CPU cores aren’t changed while conducting these experiments.

## 1.6 Thesis outline

Rest of the thesis is organized as mentioned below.

**Chapter 2:** It deals with the background knowledge, about virtualization various virtualization techniques. Different TCP congestion control mechanism.

**Chapter 3:** It gives the details of how the setup is made, how the experiment is conducted, and what all tools are used for the experiment.

**Chapter 4:** The results from the experimentation are presented in this chapter.

**Chapter 5:** The results obtained are analyzed in this chapter.

**Chapter 6:** Conclusions are drawn out from this research work, contributions, limitations and future works which can extend this work, are presented in this chapter.

## 2 RELATED WORK

An overview of different kinds of virtualization techniques such as full virtualization, para virtualization, hardware-layer virtualization and OS-layer virtualization. and which virtualization technique is better to be used in mentioned in [7]. The main reason for choosing KVM over Xen though both the hypervisors are open source software's is that, because of the complexity in installing Xen hypervisor than KVM hypervisor. The way in which the KVM hypervisor should be installed is clearly mentioned in[8] .

**Full virtualization:** In full virtualization the guest Operating system doesn't know that it's being virtualized and needs no modification. Unlike para virtualization, no modification of guest OS or application is required. The guest OS or application is not aware of the virtualized environment so they have the capability to execute on the VM just as they would on a physical system. Full virtualization can be advantageous because it enables complete separation of the software from the hardware[2].

**Para-virtualization:** Instead of virtualizing the total hardware para virtualization virtualizes part of an OS's operating environment and also emulates only the hardware devices that a virtualized OS requires. Para virtualization techniques require modifications in the guest operating systems as a result allowing near-native performance[9].

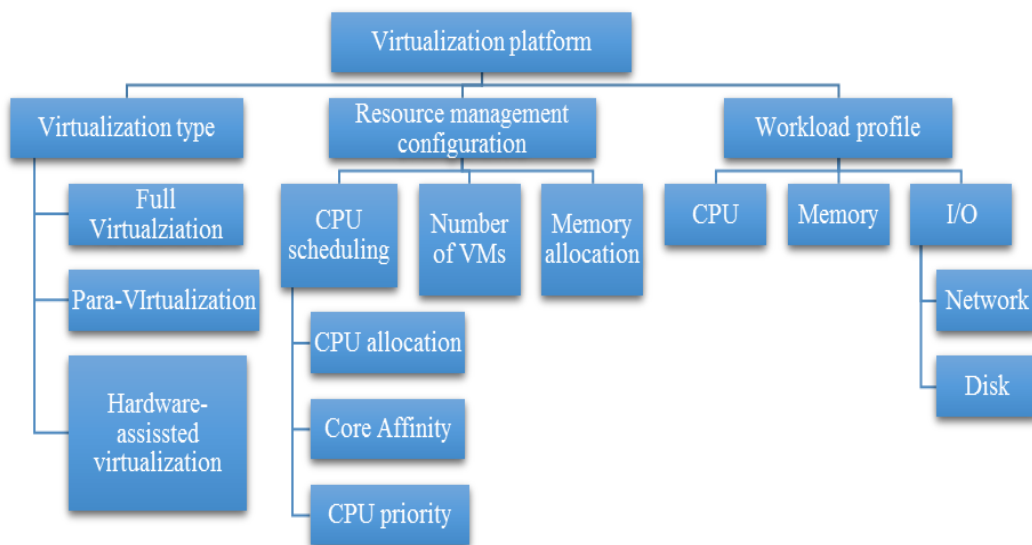


Figure 2.1: Performance influencing factors in virtualization

An overview on the research that has been made on the performance influencing factors in the virtualized environment has been studied. The research tells that the factors affecting the virtualized environment such as the type of virtualization whether it is Para virtualized or fully virtualized or binary translation. Apart from type of virtualization resource management might also influence the performance in virtualization. Resource management configurations such as number of VMs, memory management, and also CPU scheduling also affects the performance in virtualized environment which in turn is affected by CPU allocation, core affinity and CPU priority. Apart from the above provided influencing factors the other important factor affecting the performance is the type of workload on the hypervisor[10].

Having a proper congestion control algorithm between a client and a server seems to be the case in order to increase the performance of TCP. So this is how the development of the TCP congestion control had started, number of congestion control mechanisms have been developed starting from window based flow control scheme to the second version of TCP that is TCP-Tahoe and third version being TCP-Reno[11]. Since the rise of TCP Reno several other alternative congestion control algorithms have been developed some of which are mentioned below:

- **Cubic:** This congestion control algorithm modifies the linear window growth function of existing TCP standards to be a cubic function in order to improve the scalability of TCP over fast and long distance networks[12].
- **BIC:** This congestion control algorithm uses two window size control policies called additive increase and binary search increase. When the congestion window is large, additive increase with a large increment ensures square RTT unfairness as well as good scalability. Under small congestion windows, binary search increase supports TCP friendliness[13].
- **Veno:** This congestion control algorithm improves the multiplicative decrease algorithm of TCP Reno by adjusting the slow-start threshold according to the network congestion level rather than a fine-tuned drop factor and it improves the linear increase algorithm so that the connection can stay longer in an operating region in which the network bandwidth is plenarily utilized[14].
- **Westwood:** TCP Westwood controls the window utilizing end-to-end rate estimation in a way that is plenary transparent to routers and to the destination [15].
- **Vegas:** The flow control scheme of TCP Vegas is derived from TCP Reno with three modified techniques. It requires the sender to record the round trip time and time each packet is sent[11].
- **Highspeed:** HighSpeed TCP is a variation of standard TCP and thus very compatible. Like standard TCP, it utilizes packet drop inference to detect congestion [16].
- **Yeah:** Yeah TCP is another type of Highspeed TCP wherein it's got two modes one mode is fast mode and the other mode is the slow mode. In the slow mode yeah acts as TCP Reno where as in the fast mode it increases the window size according to the aggressive rule[17].
- **LP:** TCP LP, Low Priority is a distributed algorithm which uses up the excess network bandwidth. The mechanism which make TCP LP different from the other TCP congestion control algorithms is it uses one way packet delay for early congestion indications and a transparent TCP congestion avoidance policy[18].
- **Hybla:** TCP HYBLA provides a promising solution to the comprising problem of performance inconsistency in heterogeneous networks due to round trip times[19].

Each one of these TCP congestion control algorithms have been developed addressing the constraints of TCP-Reno. TCP-Cubic is the default congestion control algorithm for the Linux versions 2.6.19 to 3.1.

Different techniques are designed in order to ameliorate the performance of TCP as in virtualization a number of VM's share the CPU, the Round Trip Times do increment degrading the TCP performance. Techniques being vflood[20] , vslicer[21] and vsnoop[22] have been designed in order to amend the performance of TCP.

To mitigate the impact of latency while retaining the benefit of CPU sharing, a new class of VMs called latency-sensitive Virtual Machines, LSVMs which achieve better performance for I/O-bound applications while maintaining the same resource share as other CPU sharing VMs are introduced. LSVMs are enabled by vSlicer, a hypervisor level technique that schedules each LSVM more frequently but with a smaller micro time slice[21].

Another approach called vSnoop is introduced, where the driver domain of the host acknowledges TCP packets on behalf of the guest VMs, whenever it is safe to do so[22].

Another solution called vFlood allows a TCP sender VM to opportunistically flood the driver domain in the same host, and offloads the VM's TCP congestion control function to the driver domain in order to mask the effects of VM consolidation[20]. The main reason these techniques are not being used is because they require changes/modifications/add-ons to hypervisor and/or VM. Making it complex.

### 3 METHODOLOGY

There are many ways to evaluate a system such as experimentation with the actual system or experimentation with the model of the system. Experimentation with a model of the system can be further divided into two sub classes that is a physical model and a mathematical model. In this research a physical model has been considered the reason being, in a mathematical model different sub class mathematical models are to be considered for this research. The different sub class mathematical models being:

1. Mathematical model for the TCP/IP.
2. Mathematical model for the operating system.
3. Mathematical model for the hypervisor.
4. Mathematical model addressing the networking issues such as delay.

The mathematical model for TCP/IP being the finite state machine, FSM. The mathematical models for the other parameters might exist or might not exist and knowing about these mathematical models by a literature review and if they do not exist creating new mathematical models is beyond the scope of this thesis as it might require a huge amount of time. So keeping the time constraints in mind this research has been done on a physical model rather than a mathematical model and the measurement analysis has been done in the physical model. A mathematical model can be implemented as a future work.

#### 3.1 Experimental Methodology

In order to find out what affects the TCP performance of a VM, the allocated memory or CPU, we will execute experiments where we change the memory sizes varying between 2, 4, and 6 keeping the CPU cores constant and then in another scenario we will change the CPU cores keeping the RAM size constant. The congestion control algorithms are set to the default which is CUBIC. Only one VM is running on the hypervisor so that there would not be any competition for resource.

To find out what CCA behaves best we will change the congestion control algorithms both on the server side and the client side as depending the total download time taken we decide which congestion control algorithm will be performing the best among the different congestion control algorithms chosen.

#### 3.2 Experimental Procedure

The experimental procedure started off by setting up an experimental setup which consists of a client, measurement point, shaper and a server as shown in Figure 3.1. The link which connects both the server and the client is the test network link or the data traffic link. The main reason for introducing shaper is in order to provide fairness for some of the TCP congestion control algorithms as some of the congestion control algorithms have been developed in order to address the networking issue such as long delays. So by using the shaper delays are introduced in the link. The specifications of the devices used are mentioned in Table 3.1.

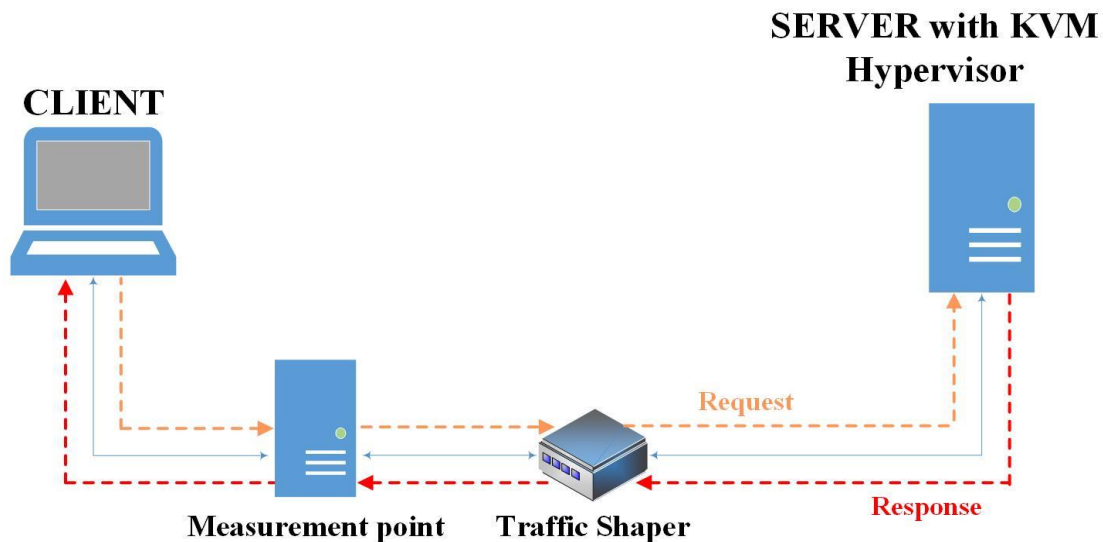


Figure 3.1: Experimental Setup

First and foremost after all the above mentioned physical connections have been made the KVM hypervisor has been installed on the server. As now the hypervisor has been installed there are different tools to install a VM over the hypervisor such as virsh, virtual-machine manager etc. In this research the virtual-machine manager has been used in order to install and run VM over the hypervisor and operate it. After installing the VM with operating system Ubuntu 13.10 over the hypervisor. Apache web server has been installed on the VM. A file having size of 1MB is stored in the /var/www folder in the VM. So, the next step is to download these files by changing the different TCP congestion control mechanisms and injecting different parameters such as delay by means of the Shaper.

**Wget** has been used instead of other file retrieval methods such as curl for downloading the files because before running the experiments a few files have been downloaded using wget and curl and it has been found out the wget used lesser time to download the file. So wget seemed to be more productive than curl as the timing is important in the experiments which are to be conducted.

The tool named as NTAS has been used to perform the experiments. The script that has been used in this research is a Perl script (provided in the appendix) that communicates with the shaper and the server. What actually happens in the Perl script is that first and foremost an SSH session is started with server and the TCP congestion control is known by the following command.

→**Sudo sysctl -n net.ipv4.tcp\_congestion\_control**

Then the congestion control is changed to the required one by the following command.

→**Sudo sysctl -w net.ipv4.tcp\_congestion\_control "desired congestion control"**

Later the same thing is carried out on the client side that is knowing the TCP congestion control and then changing it to the desired congestion control by the below commands.

→**Sudo sbin/sysctl -n net.ipv4.tcp\_congestion\_control**

→**Sudo sbin/sysctl -w net.ipv4.tcp\_congestion\_control 'desired congestion control'**

After changing the TCP congestion control a time stamp has been noted down and then the file from the server is downloaded using the wget command i.e.

→Wget http://x.x.x.x/filename

After downloading the file another time stamp had been noted down and the difference of these two time stamps will be giving us the total download time of the file. After downloading the file the TCP congestion controls are returned to their previous states. The same procedure has been repeated for 40 times. The connection to the shaper is also done through the same Perl script by starting a SSH session with the shaper and adding different delays to the shaper. After downloading the file the shaper settings are also reset to their previous values.

Apart from only changing the TCP congestion control algorithms the other parameter which effect the performance of the TCP such as delay is also injected by means of a shaper. This is how the total experiment and carried out. The specifications of the devices used are provided in the next section. The whole process of how the experiment is done is show clearly in the form of a flow chart in Figure 3.2.

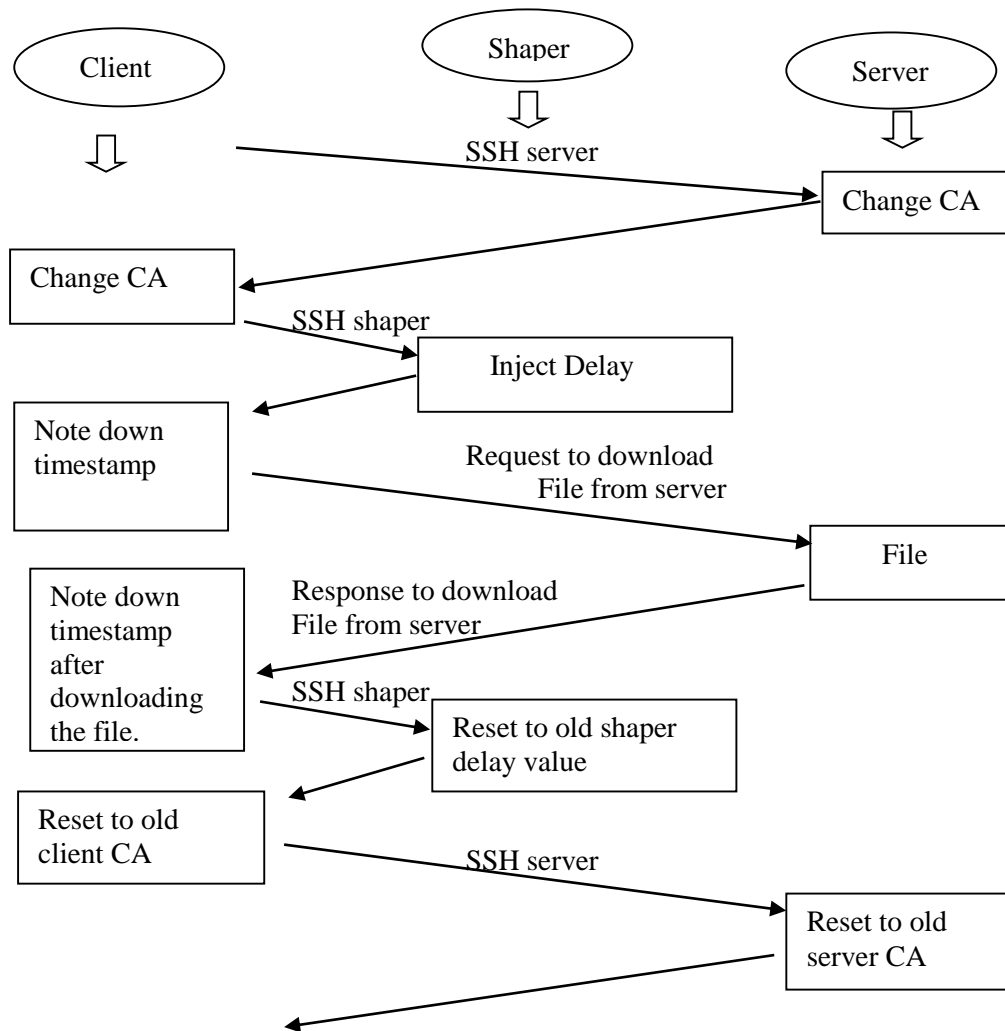


Figure 3.2: Flow chart for the Perl script

### 3.3 Hardware and Software Specifications

Name	Platform	CPU	RAM	CPU cores	OS	Hypervisor	Guest OS
Client	Dell Latitude E5500	Intel® Core™ 2	2GB	2	Ubuntu 14.04 LTS	-	-
Server	Fujitsu Primergy Rx100 S7	Inter® Xeon® CPU E3-1220 @3.10GHz*4	8GB	4	Ubuntu 13.10-Desktop	KVM	Ubuntu 13.10-server
Shaper	Dell OptiPlex 755	Inter® Core™ 2 Duo CPU E6850 @300GHz*4	2GB	2	Ubuntu 12.04 LTS	-	-

Table 3.1: Device Specifications

### 3.4 Experimental Scenarios

There were mainly two scenarios conducted in order to answer the two research questions.

#### 3.4.1 Scenario A

In the following scenario which answers the research question (a) the RAM sizes and CPU cores of the virtual machine which has been launched on the hypervisor are varied. The RAM sizes that are used are 2GB, 4GB and 6GB RAM's. The logical cores used are 1, 2 and 3. The congestion control algorithms are set to the default algorithm i.e. CUBIC. Only one VM was running on the hypervisor so that there would not be any competition for the resources. How each scenario of the experiment is done is presented neatly in the results section with the tables from which the graphs are plotted are provided in the appendix.

#### 3.4.2 Scenario B

In the following scenario different TCP congestion controls algorithms have been varied both in the client and the server in order to find out which congestion control algorithm performs the best in the virtualized environment. In this scenario the CPU and RAM sizes are kept constant 2 and 4GB respectively. The reason for taking CPU as 2 and RAM size as 4GB is that to provide equal amount of resources to both the host and the guest OS. Different experiments have been performed in this scenario which have been discussed in the results and discussion section.

## 4 RESULTS AND DISCUSSION

In order to answer the research questions the following two experiments have been conducted. The results obtained during experimentation are showcased in this section. The results are displayed using graphs. The supporting values table for graphs are given in the appendix.

### 4.1 Results from Scenario A

In the following scenario the RAM sizes and CPU cores of the virtual machine which has been launched on the hypervisor are varied. The RAM sizes that are used are 2GB, 4GB and 6GB RAM's. The number of logical CPU cores used by the VM are 1, 2 and 3. Only one VM is deployed over the hypervisor in order to avoid any competition for the resources. While varying the RAM sizes the congestion control algorithms on the server side and the client side are kept constant as CUBIC as it's the default congestion control algorithm in any linux distribution. Below are results obtained from the scenario A. Along with the change in the RAM sizes even delays of 5ms, 10ms and 20ms have been injected by means of a shaper. A total of 40 iterations have been made and the average download time from the 40 iterations have been displayed in the below tables. The reason for choosing 40 iterations is that according to the central limit theorem a minimum of 32 iterations are needed in order to calculate the confidence interval values. The respective max, min, standard deviation, 95% confidence interval, coefficient of variance values can be seen in the tables provided in the appendix.

	Memory	2GB	4GB	6GB
CPU	1	1.33	1.20	1.26
	2	1.26	1.18	1.27
	3	1.24	1.21	1.14

Table 4.1: Comparison of download times for different CPUs and RAMs

### 4.2 Results from Scenario B

In the following scenario the logical CPU core and RAM of the virtual machine which has been launched on the hypervisor are kept constant. The logical CPU cores that are used are 2 and the RAM size that has been used is 4GB. The reason for choosing 4GB is because even proportion of RAM will be allocated to both the host OS and the guest OS as the server has a total of 8GB of RAM. Similarly, with the CPU cores as the server has a total of 4 CPU cores in order to provide equal number of CPU cores between the host OS and the guest OS 2 cores have been allocate to the VM. Then the congestion control on the server side and client side are changed. The congestion control on the server side and the client side are varied. CUBIC, BIC, HIGHSPEED, VENO, VEGAS, YEAH, WESTWOOD, LP, SCALABLE, RENO, HYBLA are the TCP congestion controls that have been varied on the client side and server side and file has been downloaded from the server to client. Along with the change in the RAM sizes even delays of 5ms, 10ms and 20ms have been injected. A total of 40 iterations have been made.

The timestamps obtained when the file has been downloaded is collected into a text file and then later using MS-Excel the statistics such as mean, maximum, minimum standard deviation, 95% confidence interval, coefficient of variance have been calculated. The statistical values are shown in the appendix. Below tables show the results obtained from different cases. The average download time from the 40 iterations have been displayed in the Table 3.

			Delay=10ms								
SERVER=>	BIC	CUBIC	HIGHSPEED	VENO	VEGAS	YEAH	WESTWOOD	LP	SCALABLE	RENO	HYBLA
CLIENT	Avg(s)	Avg(s)	Avg(s)	Avg(s)	Avg(s)	Avg(s)	Avg(s)	Avg(s)	Avg(s)	Avg(s)	Avg(s)
BIC	0.67	1.15	1.66	1.39	1.21	0.74	1.61	1.67	0.93	1.69	1.60
CUBIC	0.68	1.18	1.67	1.42	1.21	0.74	1.62	1.70	0.95	1.68	1.55
HIGHSPEED	0.79	1.08	1.68	1.44	1.24	0.75	1.67	1.67	1.04	1.65	1.56
VENO	0.72	1.00	1.7	1.38	1.26	0.73	1.65	1.74	0.94	1.66	1.58
VEGAS	0.67	1.15	1.65	1.40	1.29	0.75	1.61	1.69	0.94	1.69	1.55
YEAH	0.70	1.11	1.64	1.43	1.34	0.78	1.61	1.69	0.94	1.66	1.58
WESTWOOD	0.69	1.20	1.66	1.41	1.27	0.72	1.65	1.62	0.96	1.67	1.58
LP	0.71	1.16	1.64	1.43	1.21	0.79	1.65	1.64	0.97	1.65	1.58
SCALABLE	0.71	1.16	1.64	1.42	1.32	0.72	1.68	1.67	1.02	1.63	1.55
RENO	0.74	1.18	1.65	1.40	1.28	0.73	1.61	1.64	0.95	1.68	1.57
HYBLA	0.73	1.17	1.7	1.42	1.36	0.73	1.67	1.71	0.94	1.59	1.55

Table 4.2: Comparison of download time for different CAs the delay 10ms

## 5 ANALYSIS AND DISCUSSION

From the experiments conducted to find out whether the RAM sizes and CPU cores have an effect on the performance of TCP in the virtualized environment below are the results compared for the experiments.

	Memory	2GB	4GB	6GB
CPU	1	1.33	1.20	1.26
	2	1.26	1.18	1.27
	3	1.24	1.21	1.14

Table 5.1: Comparison of download times for different CPUs and RAMs

The x-axis in the table 5.1 denotes the RAM sizes 2GB, 4GB and 6GB and the y-axis in the table denotes the CPU cores 1, 2 and 3. The delay injected is 10ms with the client congestion control algorithm and the server congestion control algorithm being the default as CUBIC.

The performance of TCP with respect to the different CPU cores i.e. 1, 2, 3 and memory sizes i.e. 2GB, 4GB, 6GB used has changed slightly. Despite running a single VM on the hypervisor and addition of resources there isn't any logical increase or decrease but a random change in the download times. But from the results obtained it is inconclusive about the performance of TCP as the download time improves and degrades with the increase of resources. There is perhaps a trend, but too vague to make a statement. In order to mitigate this problem may be increasing the sample sizes could lead to a conclusion.

The performance of TCP could be degrading because of various other factors too such as the technique used in virtualization whether it's a para virtualization or other types of virtualization or because of the type of hypervisor used in the virtualization because each hypervisor has its own merits and demerits so depending on these merits and demerits the TCP performance also might differ. Apart from that the type of bridging technique used might also change the performance of TCP. As the bridging technique used in this research is linux bridging, the research work can be carried out to other bridging techniques such as Open vSwitch or macvlan as these are the different bridging techniques that can be used in the KVM virtualized environment. So considering all these parameters further work can be done in order to know the actual cause for the performance of TCP in the virtualized environment and also the number of VMs used.

			Delay=10ms								
SERVER=>	BIC	CUBIC	HIGHSPEED	VENO	VEGAS	YEAH	WESTWOOD	LP	SCALABLE	RENO	HYBLA
CLIENT	Avg(s)	Avg(s)	Avg(s)	Avg(s)	Avg(s)	Avg(s)	Avg(s)	Avg(s)	Avg(s)	Avg(s)	Avg(s)
BIC	0.67	1.15	1.66	1.39	1.21	0.74	1.61	1.67	0.93	1.69	1.60
CUBIC	0.68	1.18	1.67	1.42	1.21	0.74	1.62	1.70	0.95	1.68	1.55
HIGHSPEED	0.79	1.08	1.68	1.44	1.24	0.75	1.67	1.67	1.04	1.65	1.56
VENO	0.72	1.00	1.7	1.38	1.26	0.73	1.65	1.74	0.94	1.66	1.58
VEGAS	0.67	1.15	1.65	1.40	1.29	0.75	1.61	1.69	0.94	1.69	1.55
YEAH	0.70	1.11	1.64	1.43	1.34	0.78	1.61	1.69	0.94	1.66	1.58
WESTWOOD	0.69	1.20	1.66	1.41	1.27	0.72	1.65	1.62	0.96	1.67	1.58
LP	0.71	1.16	1.64	1.43	1.21	0.79	1.65	1.64	0.97	1.65	1.58
SCALABLE	0.71	1.16	1.64	1.42	1.32	0.72	1.68	1.67	1.02	1.63	1.55
RENO	0.74	1.18	1.65	1.40	1.28	0.73	1.61	1.64	0.95	1.68	1.57
HYBLA	0.73	1.17	1.7	1.42	1.36	0.73	1.67	1.71	0.94	1.59	1.55

Table 5.1.2: Comparison of download times by varying different CAs

From the experiments conducted to find out which TCP congestion control algorithm performs better in the virtualized environment, after performing a number of experiments and 40 iterations per experiments it is clearly seen in Table 5. There is no variability in the delay and a constant delay is subjected to the packets passing through the shaper. Since there are a huge number of graphs and tables the minimum, maximum, standard deviation, 95% confidence interval and coefficient of variance values for the respective values are provided in the appendix. The maximum values are mainly because of the duplicate acknowledgements occurring during the communication between the client and the server. This is known by the trace files collected from the measurement point in the experimental setup. Both TCP-BIC and TCP-YEAH are the two congestion control algorithms that have performed the best in virtualized environment while downloading the time by keeping the total download time very low. These TCP congestion control algorithms performed the best when delays have been injected into the link between the server and the client by means of a shaper in order to project the real time scenario in the experimental setup. Coming to the other TCP congestion control algorithms TCP-VENO and TCP-HYBLA are the other congestion control algorithms which performed better when the delay is being increased. The performance of TCP- SCALABLE began to degrade with the increase in the delay values. TCP-LP, TCP-Reno, TCP-WESTWOOD and TCP- HYBLA have the least performance of the lot of TCP congestion control algorithms. So from these analysis we can conclude by saying that using TCP-BIC and TCP- YEAH in the virtualized environments might increase the performance of TCP by decreasing the download times.

## 6 CONCLUSION AND FUTURE WORK

In this research, in order to answer the research question (a) we present a Server-Client test-bed, with respect to the performance influencing factors in virtualization environment such as CPU cores and RAM sizes we try to make it out whether those influencing factors in virtualized environment are influencing the TCP performance. After the experiment and statistical analysis we can see that there is a slight change in the performance of the TCP with respect to these performance parameters. But from the results obtained it is inconclusive about the performance of TCP as the download time improves and degrades with the increase of resources. There is perhaps a trend, but too vague to make a statement. In order to mitigate this problem increasing the sample sizes could lead us to a conclusion.

In order to answer the research question (b) we present the better TCP congestion control that can be used in the virtualized environment in order to improve the performance of TCP in the virtualized environment. The results show that TCP-BIC and TCP-YEAH are the CA that perform the best and can be used in the virtualized environment when the delay constraints are introduced. TCP-LP and TCP-Reno are the TCP congestion control algorithms that had a low performance when the experiments are conducted and these are the congestion control algorithms that can be neglected in the virtualized environments.

Due to time constraints this research has been only limited to only two performance influencing factors in the virtualized environment they are CPU core and memory sizes. This research can be further be done on a different hypervisor and see how the performance of TCP is in that hypervisor as there are many hypervisors such as Xen, VMware, Virtual box etc. Also, the bridging used in this research is Linux bridging so the research can be extended to different kind of bridging techniques used in the hypervisor such as Open vSwitch, macvlan in KVM. Other network parameters such as Jitter, Bandwidth different file sizes can also be used in order to find out which TCP congestion control algorithm performs better in the virtualized environment under these conditions.

## REFERENCES

- [1] M. Asberg, N. Forsberg, T. Nolte, and S. Kato, "Towards real-time scheduling of virtual machines without kernel modifications," in *2011 IEEE 16th Conference on Emerging Technologies Factory Automation (ETFA)*, 2011, pp. 1–4.
- [2] T. Abels, P. Dhawan, and B. Chandrasekaran, "An overview of xen virtualization," *Dell Power Solut.*, vol. 8, pp. 109–111, 2005.
- [3] D. Marshall, "Understanding Full Virtualization, Paravirtualization, and Hardware Assist," 2007.
- [4] I. Habib, "Virtualization with KVM," *Linux J*, vol. 2008, no. 166, Feb. 2008.
- [5] "libvirt: Wiki: Virtio." [Online]. Available: <http://wiki.libvirt.org/page/Virtio>. [Accessed: 11-Aug-2015].
- [6] A. Ahuja, S. Agarwal, J. P. Singh, and R. Shorey, "Performance of TCP over different routing protocols in mobile ad-hoc networks," presented at the Vehicular Technology Conference Proceedings, 2000. VTC 2000-Spring Tokyo. 2000 IEEE 51st, 2000, vol. 3, pp. 2315–2319.
- [7] J. Sahoo, S. Mohapatra, and R. Lath, "Virtualization: A Survey on Concepts, Taxonomy and Associated Security Issues," in *2010 Second International Conference on Computer and Network Technology (ICCNT)*, 2010, pp. 222–226.
- [8] "Installing KVM on Ubuntu 14.04," *Natural Born Coder*.
- [9] S. J. Vaughan-Nichols, "New Approach to Virtualization Is a Lightweight," *Computer*, vol. 39, no. 11, pp. 12–14, Nov. 2006.
- [10] J. M. von Q. Nikolaus Huber, "Analysis of the Performance-Influencing Factors of Virtualization Platforms," pp. 811–828, 2010.
- [11] Y.-C. Lai and C.-L. Yao, "The performance comparison between TCP Reno and TCP Vegas," in *Parallel and Distributed Systems: Workshops, Seventh International Conference on, 2000*, 2000, pp. 61–66.
- [12] M. Ahmad, A. B. Ngadi, A. Nawaz, U. Ahmad, T. Mustafa, and A. Raza, "A survey on TCP CUBIC variant regarding performance," in *Multitopic Conference (INMIC), 2012 15th International*, 2012, pp. 409–412.
- [13] L. Xu, K. Harfoush, and I. Rhee, "Binary increase congestion control (BIC) for fast long-distance networks," in *INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*, 2004, vol. 4, pp. 2514–2524 vol.4.
- [14] C. P. Fu and S. C. Liew, "TCP Veno: TCP enhancement for transmission over wireless access networks," *IEEE J. Sel. Areas Commun.*, vol. 21, no. 2, pp. 216–228, Feb. 2003.
- [15] M. Gerla, M. Y. Sanadidi, R. Wang, A. Zanella, C. Casetti, and S. Mascolo, "TCP Westwood: congestion window control using bandwidth estimation," in *IEEE Global Telecommunications Conference, 2001. GLOBECOM '01*, 2001, vol. 3, pp. 1698–1702 vol.3.
- [16] L. A. Dalton and C. Isen, "A study on high speed TCP protocols," in *IEEE Global Telecommunications Conference, 2004. GLOBECOM '04*, 2004, vol. 2, pp. 851–855 Vol.2.
- [17] A. Baiocchi, A. P. Castellani, and F. Vacirca, "YeAH-TCP: yet another highspeed TCP," presented at the Proc. PFLDnet, 2007, vol. 7, pp. 37–42.
- [18] A. Kuzmanovic and E. W. Knightly, "TCP-LP: low-priority service via end-point congestion control," *IEEE ACM Trans. Netw.*, vol. 14, no. 4, pp. 739–752, Aug. 2006.
- [19] C. Caini and R. Firrincieli, "TCP Hybla: a TCP enhancement for heterogeneous networks," *Int. J. Satell. Commun. Netw.*, vol. 22, no. 5, pp. 547–566, 2004.
- [20] S. Gamage, A. Kangarlou, R. R. Kompella, and D. Xu, "Opportunistic Flooding to Improve TCP Transmit Performance in Virtualized Clouds," in *Proceedings of the 2Nd ACM Symposium on Cloud Computing*, New York, NY, USA, 2011, pp. 24:1–24:14.
- [21] C. Xu, S. Gamage, P. N. Rao, A. Kangarlou, R. R. Kompella, and D. Xu, "vSlicer: Latency-aware Virtual Machine Scheduling via Differentiated-frequency CPU Slicing," in *Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing*, New York, NY, USA, 2012, pp. 3–14.

[22] A. Kangarlou, S. Gamage, R. R. Kompella, and D. Xu, “vSnoop: Improving TCP Throughput in Virtualized Environments via Acknowledgement Offload,” in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, Washington, DC, USA, 2010, pp. 1–11.

## 7 APPENDIX

### 7.1 Experiments in Ntas

List of experiments and the command used for the experiment in ntas are given below:

```
perl          sahy14/getHttpLocalContent_ab.pl          http://192.168.186.214/1MB.txt
/mnt/LONTAS/Logs/tcpeval/          server:net.ipv4.tcp_congestion_control=hybla
client:net.ipv4.tcp_congestion_control=hybla Shaper:Delay=5 Shaper:Jitter=0 Shaper:BW=480
Shaper:Loss=0 Hypervisor:Namme=KVM VM:Mem=z VM:CPU=y
```

Experiments 2069-2099 and 2279-2280 have been run with CPU=2 and MEM=6.  
Experiments 2101-2129 and 2283-2286 have been run with CPU=2 and MEM=2.  
Experiments 2238 to 2271 have been run with CPU=3 and MEM=4.  
Experiments 2196 to 2228 have been run with CPU=1 and MEM=4.  
Experiments 2600 to 2602 have been run with CPU=1 and MEM=6.  
Experiments 2606 to 2608 have been run with CPU=3 and MEM=2.  
Experiments 2609 to 2611 have been run with CPU=3 and MEM=6.  
Experiments 2612 to 2614 have been run with CPU=1 and MEM=2.

### 7.2 Perl script used in the experiment

```
#!/usr/bin/perl

use Time::HiRes qw(usleep gettimeofday tv_interval);
use Socket;
use IO::Socket;
use Sys::Hostname;
use Switch;

$content = shift(@ARGV);

$logdir = shift(@ARGV);

printf "Expid= " . $ENV{EXPID} . " Runid= " . $ENV{RUNID} . "\n";

my $device_string=`uname -a`;
if ($device_string =~ m/Linux/i) {
    printf "Supported platform.\n";
} else {
    printf "No support for $device_string. \n";
    printf "FAILURE.\n";
    exit(0);
}
$optString="";
$tcpRESTORE="";
$tcpRESTOREserver="";

my $host = hostname();
our $shaper_port = 9000;
our $shaper_ip='10.1.0.43';
```

```

print "host = $host \n";

switch ($host) {
    case 'Linux_Desktop2' {$shaper_ip='10.1.0.74';}#1580
    case 'ubuntu' {$shaper_ip='10.1.0.52';}
    else {}
}

print Time::HiRes::time . "ShaperIP: $shaper_ip:$shaper_port \n";

my $startTime=Time::HiRes::time;

%tcpRestClient=();
%tcpRestServer=();
%shaperRestore=();

while ($option=shift(@ARGV)) {
    $location="";
    $setting="";
    if ($option =~ m/Server:/i or $option =~ m/Client:/i or $option =~ m/Shaper:/i
or $option =~ m/Hypervisor:/i or $option =~ m/VM:/i ) {
        ($location,$option)=split(':', $option);
    }
    ($setting,$newVal)=split('=', $option);
    chomp($newVal);
    printf Time::HiRes::time . " Decoded as $location, $setting, $newVal.\n";

    if ($location =~ m/Shaper/i){
        switch($setting) {
            case 'Delay' { printf "\t\tSet Delay, $newVal. "; setDelay($newVal); }
            case 'BW' { printf "\t\tSet BW, $newVal. "; setBw($newVal); }
            else {
                printf Time::HiRes::time . "\t\t Unknown setting,$setting with $newVal.
Skipping. \n";
            }
        }
    }

    if ($location =~ m/Server/i){
        printf "\t\t Executing ssh server sysctl -n $setting, ";
        $oldSetting=`ssh root@"10.1.0.146 sysctl -n $setting`;
        chomp($oldSetting);
        printf " oldValue ={$setting} - $oldSetting";
        $tcpRestServer{$setting}=$oldSetting;
    }

    if ($location =~ m/Client/i){
        printf "\t\t Executing sudo /sbin/sysctl -n $setting, ";
        $oldSetting=`sudo /sbin/sysctl -n $setting`;
        chomp($oldSetting);
        printf " oldValue = $oldSetting";
        $tcpRestClient{$setting}=$oldSetting;
    }

    printf "\n";
}

```

```

        if ($location=~ m/Server/i){
            printf "\t\t Executing ssh server sysctl -w $option ";
            }
            printf "\n";

            `ssh root"@10.1.0.146 sysctl -w $setting=\ "$newVal`";
        }
        if ($location=~ m/Client/i){
            print "\t\t Executing sudo /sbin/sysctl -w $setting='$newVal' ";

            `sudo /sbin/sysctl -w $setting='$newVal`";
        }

        $optString=sprintf("%s,%s",$optString,$option);
    }

    printf Time::HiRes::time . " Setup complete\n";

    my($scheme, $contentIP, $path, $query, $fragment) = $content =~
m(?:([^\./?#]+):)?(?://(?:[^\./?#]*)(?:\?([^\#]*)|(?:(#(?:.))*))?)?;

    printf Time::HiRes::time . " Verifying shaping: ";
    $pingresult=`ping -i 0.2 -c 5 $contentIP | grep -E 'transmitted|rtt' | tr '\n' ' '`;
    printf " $pingresult .\n";

    local $| = 1;
    my $host = hostname();
    my $ipaddr = inet_ntoa(scalar gethostbyname($host || 'localhost'));

    print MyLOG Time::HiRes::time . " device known as $host with $ipaddr: ";
    $tStart=Time::HiRes::time;
    printf Time::HiRes::time . "Will execute: \n";
    my $wget=`sudo wget $content `;
    $tEn=Time::HiRes::time;
    my $wrm=`sudo rm 1MB.txt`;
    open( FP,">>/mnt/LONTAS/ExpControl/sahy14/results/$ENV{EXPID}.txt") or
die $1;

    print FP ($tEn-$tStart)."\n";
    printf "download time".($tStart-$tEn)."\n";

    printf Time::HiRes::time . " Resetting.\n";
    printf Time::HiRes::time . " Restoring Server .\n";
    for $setting (keys %tcpRestServer) {
        $oldSetting=$tcpRestServer{$setting};
        chomp($oldSetting);
        printf "\t\t Executing ssh server sysctl -w $setting=\ "$oldSetting`"\n";
        `ssh root"@10.1.0.146 sysctl -w $setting=\ "$oldSetting`";
    }

```

```

}

printf Time::HiRes::time . " Restoring Client .\n";
for $setting (keys %tcpRestClient) {
    $oldSetting=$tcpRestClient{$setting};
    chomp($oldSetting);
    printf "\t\t Executing /sbin/sysctl -w $setting=\""$oldSetting "\"\n";
    `sudo /sbin/sysctl -w $setting=\""$oldSetting "\"`;
}

printf Time::HiRes::time . " Restoring Shaper .\n";
cleanNetem();
printf Time::HiRes::time . " Verifying shaping: ";
$pingresult=`ping -i 0.2 -c 5 $contentIP | grep -E 'transmitted|rtt' | tr '\n' ' '`;
printf " $pingresult .\n";

close(MyLOG);

my $endTime=Time::HiRes::time;
print "SUCCESS\n";
exit;

sub setDelay{
    $sProcess = 'setDelay';
    my $socket = Shaper_manager();
    $socket->send("setDelay:".$_[0]."\n");
    $socket->recv($rx_buf,128);
    printf "\tShaper return: $rx_buf ";
    close($socket);
    if(foundERR($rx_buf)){
        return 0;
    }else{
        return 1;
    }
}

sub setBw{
    $sProcess = 'setBw';
    my $socket = Shaper_manager();
    $socket->send("setBw:".$_[0]."\n");
    $socket->recv($rx_buf,128);
    printf "\tShaper return: $rx_buf ";
    close($socket);
    if(foundERR($rx_buf)){
        return 0;
    }else{
        return 1;
    }
}

sub cleanNetem{
    $sProcess = 'cleanNetem';
    my $socket = Shaper_manager();
    $socket->send("cleanNetem:\n");
    $socket->recv($rx_buf,128);
    printf "\tShaper return: $rx_buf";
}

```

```

close($socket);
if(foundERR($rx_buf)){
    return 1;
}else{
    return 0;
}
}

sub Shaper_manager{
    # Setting for Shaper manager
    my $IP = $shaper_ip;
    my $PORT = $shaper_port;
    printf Time::HiRes::time . "Shaper, $IP - $PORT \n";
    my $socket = new IO::Socket::INET (PeerAddr => $IP, PeerPort => $PORT,
Proto => 'tcp');
    die printf Time::HiRes::time . "Shaper return: $rx_buf \n" unless $socket;
    return $socket;
}

sub foundERR{
    if ($_[0] =~ /abort/){
        print "CRAP:player abort, press any key to continue.\n";
    }
    if ($_[0] =~ /EE/){
        print "EE found.";
    }
    if ($_[0] =~ /ERR/){
        print "ERR found in process: $sProcess.\n";
        return 1;
    }else{
        return 0;
    }
}
}

```

### 7.3 Numerical values obtained for Scenario A

SERVER	CUBIC	Memory	Delay	5ms
CLIENT	CUBIC	2GB	4GB	6GB
	<b>1</b>	0.75	0.67	0.79
<b>CPU</b>	<b>2</b>	0.75	0.69	0.72
	<b>3</b>	0.68	0.71	0.62

Table 7.1: Comparison of download times by varying CPUs and RAM for 5ms delay

SERVER	CUBIC	Memory	Delay	10ms
CLIENT	CUBIC	2GB	4GB	6GB
	<b>1</b>	1.33	1.20	1.26
<b>CPU</b>	<b>2</b>	1.26	1.18	1.27
	<b>3</b>	1.24	1.21	1.14

Table 7.2: Comparison of download times by varying CPUs and RAM for 10ms delay

SERVER	CUBIC	Memory	Delay	20ms	
CLIENT	CUBIC	2GB	4GB	6GB	
		1	2.10	2.04	1.94
<b>CPU</b>		2	1.93	1.70	1.94
		3	1.93	1.87	1.96

Table 7.3: Comparison of download times by varying CPUs and RAM for 20ms delay

SERVER=CUBIC	Delay=5ms					
CLIENT=CUBIC	Avg	Max	Min	Stdev	95% CI	COV
<b>CPU=1 RAM=2</b>	0.75	1.07	0.39	0.17	0.05	0.23
<b>CPU=1 RAM=4</b>	0.67	0.97	0.46	0.15	0.04	0.22
<b>CPU=1 RAM=6</b>	0.79	0.94	0.40	0.15	0.05	0.19
<b>CPU=2 RAM=2</b>	0.75	0.93	0.48	0.15	0.05	0.19
<b>CPU=2 RAM=4</b>	0.69	0.93	0.49	0.15	0.05	0.22
<b>CPU=2 RAM=6</b>	0.72	0.96	0.50	0.15	0.05	0.20
<b>CPU=3 RAM=2</b>	0.68	0.92	0.37	0.19	0.06	0.28
<b>CPU=3 RAM=4</b>	0.71	0.91	0.51	0.14	0.04	0.19
<b>CPU=3 RAM=6</b>	0.62	1.33	0.37	0.19	0.06	0.31

Table 7.4: Statistical analysis of different CPU and RAM values for delay 5ms

SERVER=CUBIC	Delay=10ms					
CLIENT=CUBIC	Avg	Max	Min	Stdev	95% CI	COV
<b>CPU=1 RAM=2</b>	1.33	1.66	0.59	0.29	0.09	0.22
<b>CPU=1 RAM=4</b>	1.20	1.57	0.78	0.27	0.08	0.23
<b>CPU=1 RAM=6</b>	1.26	1.73	0.63	0.31	0.10	0.25
<b>CPU=2 RAM=2</b>	1.26	1.77	0.59	0.34	0.11	0.27
<b>CPU=2 RAM=4</b>	1.18	2.09	0.55	0.42	0.13	0.35
<b>CPU=2 RAM=6</b>	1.27	1.60	0.78	0.26	0.08	0.21
<b>CPU=3 RAM=2</b>	1.24	1.72	0.58	0.30	0.09	0.24
<b>CPU=3 RAM=4</b>	1.21	1.56	0.82	0.26	0.08	0.21
<b>CPU=3 RAM=6</b>	1.14	1.55	0.55	0.31	0.10	0.27

Table 7.5: Statistical analysis of different CPU and RAM values for delay 10ms

SERVER=CUBIC	Delay=20ms					
CLIENT=CUBIC	Avg	Max	Min	Stdev	95% CI	COV
<b>CPU=1 RAM=2</b>	2.10	2.92	1.26	0.55	0.17	0.26
<b>CPU=1 RAM=4</b>	2.04	2.92	1.24	0.56	0.17	0.27
<b>CPU=1 RAM=6</b>	1.94	3.01	0.90	0.58	0.18	0.30
<b>CPU=2 RAM=2</b>	1.93	2.81	1.13	0.55	0.17	0.29
<b>CPU=2 RAM=4</b>	1.70	3.04	0.84	0.51	0.16	0.30
<b>CPU=2 RAM=6</b>	1.94	3.10	0.80	0.63	0.19	0.32
<b>CPU=3 RAM=2</b>	1.93	2.92	0.85	0.58	0.18	0.30
<b>CPU=3 RAM=4</b>	1.87	3.08	1.07	0.59	0.18	0.32
<b>CPU=3 RAM=6</b>	1.96	2.78	0.80	0.61	0.19	0.31

Table 7.6: Statistical analysis of different CPU and RAM values for delay 20ms

## 7.4 Numerical values obtained for Scenario B

The numerical values obtained from the different scenarios conducted are given below. The confidence interval calculated is for 95%. Abbreviated COV is Coefficient of Variance. The denotation of seconds is abbreviated with small letter (s).

	Delay=5ms										
SERVER->	BIC	CUBIC	HIGHSPEED	VENO	VEGAS	YEAH	WESTWOOD LP	SCALABLE	RENO	HYBLA	
CLIENT	Average(s)	Average(s)	Average(s)	Average(s)	Average(s)	Average(s)	Average(s)	Average(s)	Average(s)	Average(s)	Average(s)
BIC	0.49	0.75	0.99	0.84	0.76	0.54	0.96	1.16	0.63	0.94	0.91
CUBIC	0.45	0.69	1.03	0.82	0.73	0.54	0.98	1.09	0.73	0.99	0.91
HIGHSPEED	0.46	0.69	1.01	0.84	0.72	0.54	0.98	1.08	0.65	0.97	0.91
VENO	0.44	0.61	1.12	0.82	0.68	0.54	0.98	1.03	0.62	0.98	0.92
VEGAS	0.49	0.73	1	0.81	0.73	0.53	0.98	1.06	0.63	0.97	0.90
YEAH	0.47	0.64	1.07	0.81	0.74	0.55	0.90	1.11	0.66	0.96	0.91
WESTWOOD	0.47	0.61	1.07	0.87	0.69	0.55	1.01	1.17	0.62	0.96	0.92
LP	0.50	0.65	1.06	0.84	0.73	0.55	0.96	1.13	0.61	0.98	0.91
SCALABLE	0.47	0.66	1.02	0.84	0.84	0.55	0.99	1.13	0.64	0.98	0.91
RENO	0.47	0.67	1.08	0.85	0.81	0.56	0.93	1.12	0.64	1.00	0.93
HYBLA	0.49	0.62	1.12	0.84	0.84	0.54	0.95	1.12	0.61	0.96	0.91

Table 7.7: Comparison of download time for different CAs the delay 5ms

	Delay=10ms										
SERVER->	BIC	CUBIC	HIGHSPEED	VENO	VEGAS	YEAH	WESTWOOD LP	SCALABLE	RENO	HYBLA	
CLIENT	Average(s)	Average(s)	Average(s)	Average(s)	Average(s)	Average(s)	Average(s)	Average(s)	Average(s)	Average(s)	Average(s)
BIC	0.67	1.15	1.66	1.39	1.21	0.74	1.61	1.67	0.93	1.69	1.60
CUBIC	0.68	1.18	1.67	1.42	1.21	0.74	1.62	1.70	0.95	1.68	1.55
HIGHSPEED	0.79	1.08	1.68	1.44	1.24	0.75	1.67	1.67	1.04	1.65	1.56
VENO	0.72	1.00	1.7	1.38	1.26	0.73	1.65	1.74	0.94	1.66	1.58
VEGAS	0.67	1.15	1.65	1.40	1.29	0.75	1.61	1.69	0.94	1.69	1.55
YEAH	0.70	1.11	1.64	1.43	1.34	0.78	1.61	1.69	0.94	1.66	1.58
WESTWOOD	0.69	1.20	1.66	1.41	1.27	0.72	1.65	1.62	0.96	1.67	1.58
LP	0.71	1.16	1.64	1.43	1.21	0.79	1.65	1.64	0.97	1.65	1.58
SCALABLE	0.71	1.16	1.64	1.42	1.32	0.72	1.68	1.67	1.02	1.63	1.55
RENO	0.74	1.18	1.65	1.40	1.28	0.73	1.61	1.64	0.95	1.68	1.57
HYBLA	0.73	1.17	1.7	1.42	1.36	0.73	1.67	1.71	0.94	1.59	1.55

Table 7.8: Comparison of download time for different CAs the delay 10ms

	Delay=20ms										
SERVER->	BIC	CUBIC	HIGHSPEED	VENO	VEGAS	YEAH	WESTWOOD	LP	SCALABLE	RENO	HYBLA
CLIENT	Average(s)	Average(s)	Average(s)	Average(s)	Average(s)	Average(s)	Average(s)	Average(s)	Average(s)	Average(s)	Average(s)
BIC	1.17	1.71	2.79	2.35	1.96	1.22	2.71	3.03	1.70	2.73	2.25
CUBIC	1.21	1.70	2.8	2.41	1.89	1.21	2.75	3.08	1.60	2.63	2.21
HIGHSPEED	1.18	1.92	2.74	2.40	1.79	1.25	2.84	3.02	1.58	2.69	2.33
VENO	1.20	2.11	2.61	2.34	1.80	1.21	2.76	3.04	1.70	2.69	2.29
VEGAS	1.13	1.83	2.69	2.39	1.89	1.27	2.84	3.14	1.57	2.81	2.25
YEAH	1.15	1.86	2.82	2.47	1.87	1.24	2.80	3.02	1.64	2.73	2.23
WESTWOOD	1.21	1.82	2.85	2.37	1.87	1.24	2.76	3.10	1.76	2.72	2.36
LP	1.23	1.98	2.9	2.35	1.80	1.25	2.83	3.12	1.68	2.77	2.40
SCALABLE	1.19	1.95	2.92	2.32	1.88	1.18	2.79	3.27	1.67	2.65	2.25
RENO	1.27	1.79	2.99	2.36	1.95	1.32	2.79	2.90	1.63	2.71	2.19
HYBLA	1.18	2.03	2.98	2.27	1.82	1.27	2.72	3.10	1.72	2.72	2.30

Table 7.9: Comparison of download time for different CAs the delay 20ms

	SERVER	CUBIC	DELAY	5ms			
	Average(s)	Max(s)	Min(s)	Stdev(s)	95% CI(s)	COV	
<b>BIC</b>	0.75	1.06	0.51	0.16	0.05	0.21	
<b>CUBIC</b>	0.69	0.93	0.49	0.15	0.05	0.22	
<b>HIGHSPEED</b>	0.69	0.95	0.50	0.15	0.05	0.22	
<b>VENO</b>	0.61	0.90	0.46	0.12	0.04	0.20	
<b>VEGAS</b>	0.73	0.92	0.50	0.15	0.05	0.21	
<b>YEAH</b>	0.64	0.89	0.47	0.13	0.04	0.21	
<b>WESTWOOD</b>	0.61	0.85	0.47	0.10	0.03	0.16	
<b>LP</b>	0.65	0.93	0.47	0.14	0.04	0.21	
<b>SCALABLE</b>	0.66	0.93	0.48	0.14	0.04	0.21	
<b>RENO</b>	0.67	0.92	0.47	0.14	0.04	0.21	
<b>HYBLA</b>	0.62	0.93	0.46	0.13	0.04	0.21	

	SERVER	CUBIC	DELAY	10ms			
	Average(s)	Max(s)	Min(s)	Stdev(s)	95% CI(s)	COV	
<b>BIC</b>	1.15	1.64	0.55	0.38	0.12	0.33	
<b>CUBIC</b>	1.18	2.09	0.55	0.42	0.13	0.35	
<b>HIGHSPEED</b>	1.08	1.56	0.57	0.36	0.11	0.33	
<b>VENO</b>	1.00	1.63	0.55	0.31	0.10	0.31	
<b>VEGAS</b>	1.15	1.58	0.55	0.34	0.10	0.29	
<b>YEAH</b>	1.11	1.88	0.55	0.39	0.12	0.35	
<b>WESTWOOD</b>	1.20	1.80	0.53	0.37	0.11	0.31	
<b>LP</b>	1.16	2.06	0.55	0.41	0.13	0.35	
<b>SCALABLE</b>	1.16	1.62	0.57	0.35	0.11	0.30	
<b>RENO</b>	1.18	1.62	0.57	0.32	0.10	0.27	
<b>HYBLA</b>	1.17	1.70	0.55	0.36	0.11	0.31	

	SERVER	CUBIC	DELAY	20ms		
	Average(s)	Max(s)	Min(s)	Stdev(s)	95% CI(s)	COV
<b>BIC</b>	1.89	2.82	1.09	0.57	0.18	0.30
<b>CUBIC</b>	1.94	3.10	0.80	0.63	0.19	0.32
<b>HIGHSPEED</b>	1.73	2.76	1.10	0.43	0.13	0.25
<b>VENO</b>	1.74	2.82	1.18	0.46	0.14	0.26
<b>VEGAS</b>	1.69	2.94	1.04	0.49	0.15	0.29
<b>YEAH</b>	1.94	2.93	1.09	0.57	0.18	0.29
<b>WESTWOOD</b>	1.62	3.01	1.10	0.48	0.15	0.30
<b>LP</b>	1.73	2.83	1.08	0.48	0.15	0.28
<b>RENO</b>	2.00	3.16	1.20	0.57	0.18	0.28
<b>SCALABLE</b>	2.03	2.82	1.21	0.54	0.17	0.27
<b>HYBLA</b>	1.74	3.00	1.09	0.52	0.16	0.30

Table 7.10: Values for CUBIC vs other TCP congestion algorithms

	SERVER	BIC	DELAY	5ms		
	Average(s)	Max(s)	Min(s)	Stdev(s)	95% CI(s)	COV
<b>BIC</b>	0.49	1.11	0.39	0.12	0.04	0.25
<b>CUBIC</b>	0.45	1.11	0.38	0.12	0.04	0.26
<b>HIGHSPEED</b>	0.46	0.80	0.38	0.10	0.03	0.22
<b>VENO</b>	0.44	0.60	0.38	0.06	0.02	0.13
<b>VEGAS</b>	0.49	0.80	0.38	0.11	0.03	0.22
<b>YEAH</b>	0.47	0.85	0.38	0.10	0.03	0.21
<b>WESTWOOD</b>	0.47	1.00	0.38	0.11	0.03	0.24
<b>LP</b>	0.50	1.38	0.38	0.19	0.06	0.38
<b>SCALABLE</b>	0.47	0.95	0.37	0.12	0.04	0.27
<b>RENO</b>	0.47	0.86	0.37	0.10	0.03	0.22
<b>HYBLA</b>	0.49	1.41	0.38	0.19	0.06	0.38

	SERVER	BIC	DELAY	10ms		
	Average(s)	Max(s)	Min(s)	Stdev(s)	95% CI(s)	COV
<b>BIC</b>	0.67	0.99	0.53	0.12	0.04	0.18
<b>CUBIC</b>	0.68	1.01	0.53	0.13	0.04	0.19
<b>HIGHSPEED</b>	0.79	1.57	0.53	0.31	0.10	0.40
<b>VENO</b>	0.72	1.47	0.53	0.20	0.06	0.28
<b>VEGAS</b>	0.67	0.97	0.53	0.12	0.04	0.18
<b>YEAH</b>	0.70	1.17	0.53	0.15	0.05	0.22
<b>WESTWOOD</b>	0.69	1.31	0.53	0.19	0.06	0.28
<b>LP</b>	0.71	1.73	0.53	0.21	0.06	0.29
<b>SCALABLE</b>	0.71	1.30	0.55	0.17	0.05	0.24
<b>RENO</b>	0.74	1.24	0.53	0.18	0.06	0.25
<b>HYBLA</b>	0.73	1.17	0.55	0.17	0.05	0.23

	<b>SERVER</b>	<b>BIC</b>	<b>DELAY</b>	<b>20ms</b>		
	<b>Average(s)</b>	<b>Max(s)</b>	<b>Min(s)</b>	<b>Stdev(s)</b>	<b>95% CI(s)</b>	<b>COV</b>
<b>BIC</b>	1.17	2.42	0.76	0.39	0.12	0.33
<b>CUBIC</b>	1.21	2.59	0.78	0.45	0.14	0.37
<b>HIGHSPEED</b>	1.18	2.82	0.84	0.36	0.11	0.31
<b>VENO</b>	1.20	2.25	0.80	0.33	0.10	0.28
<b>VEGAS</b>	1.13	1.78	0.77	0.30	0.09	0.26
<b>YEAH</b>	1.15	1.92	0.77	0.29	0.09	0.25
<b>WESTWOOD</b>	1.21	2.02	0.78	0.32	0.10	0.26
<b>LP</b>	1.23	2.42	0.79	0.46	0.14	0.38
<b>SCALABLE</b>	1.19	2.39	0.81	0.33	0.10	0.28
<b>RENO</b>	1.27	2.49	0.82	0.46	0.14	0.37
<b>HYBLA</b>	1.18	2.20	0.76	0.32	0.10	0.27

Table 7.11: Values for BIC vs other TCP congestion algorithms

	<b>SERVER</b>	<b>VENO</b>	<b>DELAY</b>	<b>5ms</b>		
	<b>Average(s)</b>	<b>Max(s)</b>	<b>Min(s)</b>	<b>Stdev(s)</b>	<b>95% CI(s)</b>	<b>COV</b>
<b>BIC</b>	0.84	1.07	0.68	0.09	0.03	0.11
<b>CUBIC</b>	0.82	1.08	0.64	0.08	0.02	0.09
<b>HIGHSPEED</b>	0.84	1.13	0.70	0.07	0.02	0.08
<b>VENO</b>	0.82	1.08	0.65	0.07	0.02	0.08
<b>VEGAS</b>	0.81	1.07	0.69	0.07	0.02	0.09
<b>YEAH</b>	0.81	0.88	0.64	0.05	0.02	0.07
<b>WESTWOOD</b>	0.87	1.08	0.76	0.08	0.03	0.10
<b>LP</b>	0.84	1.11	0.67	0.09	0.03	0.11
<b>RENO</b>	0.84	1.09	0.73	0.10	0.03	0.11
<b>SCALABLE</b>	0.85	1.27	0.69	0.08	0.02	0.09
<b>HYBLA</b>	0.84	1.21	0.71	0.08	0.03	0.10

	<b>SERVER</b>	<b>VENO</b>	<b>DELAY</b>	<b>10ms</b>		
	<b>Average(s)</b>	<b>Max(s)</b>	<b>Min(s)</b>	<b>Stdev(s)</b>	<b>95% CI(s)</b>	<b>COV</b>
<b>BIC</b>	1.39	1.51	1.12	0.09	0.03	0.06
<b>CUBIC</b>	1.42	1.69	1.16	0.11	0.03	0.08
<b>HIGHSPEED</b>	1.44	1.69	1.16	0.11	0.04	0.08
<b>VENO</b>	1.38	1.68	1.11	0.11	0.03	0.08
<b>VEGAS</b>	1.40	1.64	1.13	0.12	0.04	0.09
<b>YEAH</b>	1.43	1.82	1.24	0.10	0.03	0.07
<b>WESTWOOD</b>	1.41	1.67	1.09	0.12	0.04	0.08
<b>LP</b>	1.43	1.69	1.17	0.08	0.03	0.06
<b>RENO</b>	1.42	1.79	1.09	0.14	0.05	0.10
<b>SCALABLE</b>	1.40	1.71	1.14	0.12	0.04	0.09
<b>HYBLA</b>	1.42	1.67	1.21	0.08	0.03	0.06

	SERVER	VENO	DELAY	20ms		
	Average(s)	Max(s)	Min(s)	Stdev(s)	95% CI(s)	COV
BIC	2.35	3.05	1.79	0.27	0.09	0.12
CUBIC	2.41	2.74	2.05	0.21	0.07	0.09
HIGHSPEED	2.40	2.92	1.95	0.27	0.09	0.11
VENO	2.34	2.69	1.76	0.25	0.08	0.11
VEGAS	2.39	2.76	1.95	0.24	0.08	0.10
YEAH	2.47	2.92	1.87	0.23	0.07	0.09
WESTWOOD	2.37	2.76	1.88	0.21	0.07	0.09
LP	2.35	3.00	1.75	0.29	0.09	0.12
RENO	2.32	2.79	1.74	0.25	0.08	0.11
SCALABLE	2.36	2.79	1.71	0.24	0.08	0.10
HYBLA	2.27	2.76	1.63	0.29	0.09	0.13

Table 7.12: Values for VENO vs other TCP congestion algorithms

	SERVER	VEGAS	DELAY	5ms		
	Average(s)	Max(s)	Min(s)	Stdev(s)	95% CI(s)	COV
BIC	0.76	1.18	0.55	0.15	0.05	0.19
CUBIC	0.73	1.17	0.51	0.16	0.05	0.21
HIGHSPEED	0.72	1.12	0.54	0.15	0.05	0.21
VENO	0.68	1.08	0.52	0.14	0.04	0.20
VEGAS	0.73	1.18	0.55	0.16	0.05	0.22
YEAH	0.74	1.12	0.53	0.16	0.05	0.22
WESTWOOD	0.69	1.34	0.53	0.17	0.06	0.25
LP	0.73	1.07	0.56	0.15	0.05	0.21
RENO	0.84	1.22	0.58	0.14	0.04	0.16
SCALABLE	0.81	1.16	0.55	0.15	0.05	0.18
HYBLA	0.84	1.18	0.52	0.16	0.05	0.19

	SERVER	VEGAS	DELAY	10ms		
	Average(s)	Max(s)	Min(s)	Stdev(s)	95% CI(s)	COV
BIC	1.21	1.72	0.82	0.25	0.08	0.21
CUBIC	1.21	1.79	0.80	0.23	0.07	0.19
HIGHSPEED	1.24	1.77	0.84	0.25	0.08	0.20
VENO	1.26	1.92	0.80	0.30	0.10	0.24
VEGAS	1.29	2.12	0.84	0.29	0.09	0.22
YEAH	1.34	1.97	0.84	0.27	0.09	0.20
WESTWOOD	1.27	2.04	0.82	0.28	0.09	0.22
LP	1.21	1.90	0.78	0.26	0.08	0.22
RENO	1.32	1.87	0.80	0.27	0.09	0.21
SCALABLE	1.28	1.84	0.82	0.27	0.08	0.21
HYBLA	1.36	2.07	0.82	0.25	0.08	0.19

	<b>SERVER</b>	<b>VEGAS</b>	<b>DELAY</b>	<b>20ms</b>		
	<b>Average(s)</b>	<b>Max(s)</b>	<b>Min(s)</b>	<b>Stdev(s)</b>	<b>95% CI(s)</b>	<b>COV</b>
<b>BIC</b>	1.96	3.07	1.34	0.48	0.15	0.24
<b>CUBIC</b>	1.89	3.14	1.12	0.48	0.15	0.26
<b>HIGHSPEED</b>	1.79	2.63	1.14	0.36	0.12	0.20
<b>VENO</b>	1.80	2.76	1.17	0.46	0.15	0.26
<b>VEGAS</b>	1.89	3.08	1.13	0.45	0.14	0.24
<b>YEAH</b>	1.87	3.14	1.26	0.49	0.16	0.26
<b>WESTWOOD</b>	1.87	2.87	1.23	0.44	0.14	0.24
<b>LP</b>	1.80	2.79	1.18	0.43	0.14	0.24
<b>RENO</b>	1.88	3.05	1.17	0.40	0.13	0.21
<b>SCALABLE</b>	1.95	3.43	1.27	0.48	0.15	0.24
<b>HYBLA</b>	1.82	2.52	1.15	0.37	0.12	0.20

Table 7.13: Values for VEGAS vs other TCP congestion algorithms

	<b>SERVER</b>	YEAH	<b>DELAY</b>	5ms		
	<b>Average(s)</b>	<b>Max(s)</b>	<b>Min(s)</b>	<b>Stdev(s)</b>	<b>95% CI(s)</b>	<b>COV</b>
<b>BIC</b>	0.54	0.84	0.48	0.07	0.02	0.12
<b>CUBIC</b>	0.54	0.88	0.47	0.08	0.03	0.15
<b>HIGHSPEED</b>	0.54	1.09	0.48	0.10	0.03	0.18
<b>VENO</b>	0.54	0.79	0.45	0.06	0.02	0.12
<b>VEGAS</b>	0.53	0.75	0.46	0.06	0.02	0.11
<b>YEAH</b>	0.55	0.79	0.45	0.08	0.03	0.15
<b>WESTWOOD</b>	0.55	0.75	0.48	0.07	0.02	0.13
<b>LP</b>	0.55	0.76	0.48	0.07	0.02	0.13
<b>RENO</b>	0.55	0.85	0.48	0.08	0.02	0.14
<b>SCALABLE</b>	0.56	0.95	0.46	0.12	0.04	0.21
<b>HYBLA</b>	0.54	0.67	0.48	0.03	0.01	0.06
	<b>SERVER</b>	YEAH	<b>DELAY</b>	10ms		
	<b>Average(s)</b>	<b>Max(s)</b>	<b>Min(s)</b>	<b>Stdev(s)</b>	<b>95% CI(s)</b>	<b>COV</b>
<b>BIC</b>	0.74	0.88	0.66	0.05	0.02	0.07
<b>CUBIC</b>	0.74	1.14	0.67	0.07	0.02	0.10
<b>HIGHSPEED</b>	0.75	1.04	0.68	0.08	0.03	0.11
<b>VENO</b>	0.73	0.84	0.65	0.04	0.01	0.05
<b>VEGAS</b>	0.75	1.08	0.66	0.09	0.03	0.13
<b>YEAH</b>	0.78	1.31	0.67	0.14	0.04	0.18
<b>WESTWOOD</b>	0.72	0.95	0.66	0.05	0.01	0.06
<b>LP</b>	0.79	1.36	0.68	0.14	0.04	0.18
<b>RENO</b>	0.72	0.84	0.67	0.04	0.01	0.05
<b>SCALABLE</b>	0.73	0.79	0.65	0.04	0.01	0.05
<b>HYBLA</b>	0.73	0.94	0.66	0.05	0.02	0.07
	<b>SERVER</b>	YEAH	<b>DELAY</b>	20ms		
	<b>Average(s)</b>	<b>Max(s)</b>	<b>Min(s)</b>	<b>Stdev(s)</b>	<b>95% CI(s)</b>	<b>COV</b>
<b>BIC</b>	1.22	1.64	0.97	0.14	0.04	0.11
<b>CUBIC</b>	1.21	1.58	1.02	0.12	0.04	0.10
<b>HIGHSPEED</b>	1.25	1.83	1.02	0.18	0.06	0.14
<b>VENO</b>	1.21	1.52	1.02	0.10	0.03	0.08
<b>VEGAS</b>	1.27	1.86	1.01	0.15	0.05	0.12
<b>YEAH</b>	1.24	1.61	1.00	0.14	0.04	0.11
<b>WESTWOOD</b>	1.24	1.70	1.08	0.13	0.04	0.10
<b>LP</b>	1.25	1.65	1.00	0.14	0.04	0.11
<b>RENO</b>	1.18	1.43	1.03	0.09	0.03	0.08
<b>SCALABLE</b>	1.32	1.80	1.06	0.18	0.06	0.14
<b>HYBLA</b>	1.27	1.69	1.08	0.11	0.04	0.09

Table 7.14: Values for YEAH vs other TCP congestion algorithms

	<b>SERVER</b>	<b>WESTWOOD</b>	<b>DELAY</b>	5ms			
	<b>Average(s)</b>	<b>Max(s)</b>	<b>Min(s)</b>	<b>Stdev(s)</b>	<b>95% CI(s)</b>	<b>COV</b>	
<b>BIC</b>	0.96	1.18	0.71	0.10	0.03	0.10	
<b>CUBIC</b>	0.98	1.23	0.78	0.07	0.02	0.07	
<b>HIGHSPEED</b>	0.98	1.23	0.78	0.11	0.04	0.12	
<b>VENO</b>	0.98	1.20	0.83	0.11	0.04	0.11	
<b>VEGAS</b>	0.98	1.24	0.78	0.13	0.04	0.14	
<b>YEAH</b>	0.90	0.99	0.76	0.05	0.01	0.05	
<b>WESTWOOD</b>	1.01	1.32	0.77	0.13	0.04	0.13	
<b>LP</b>	0.96	1.21	0.74	0.13	0.04	0.13	
<b>SCALABLE</b>	0.99	1.19	0.76	0.11	0.03	0.11	
<b>RENO</b>	0.93	1.22	0.72	0.09	0.03	0.10	
<b>HYBLA</b>	0.95	1.16	0.78	0.10	0.03	0.11	
	<b>SERVER</b>	<b>WESTWOOD</b>	<b>DELAY</b>	10ms			
	<b>Average(s)</b>	<b>Max(s)</b>	<b>Min(s)</b>	<b>Stdev(s)</b>	<b>95% CI(s)</b>	<b>COV</b>	
<b>BIC</b>	1.61	1.91	1.22	0.14	0.04	0.08	
<b>CUBIC</b>	1.62	1.95	1.24	0.18	0.06	0.11	
<b>HIGHSPEED</b>	1.67	1.97	1.24	0.17	0.06	0.10	
<b>VENO</b>	1.65	2.31	1.22	0.19	0.06	0.11	
<b>VEGAS</b>	1.61	1.94	1.34	0.13	0.04	0.08	
<b>YEAH</b>	1.61	1.89	1.13	0.15	0.05	0.09	
<b>WESTWOOD</b>	1.65	1.98	1.36	0.15	0.05	0.09	
<b>LP</b>	1.65	1.96	1.24	0.12	0.04	0.07	
<b>SCALABLE</b>	1.68	1.94	1.22	0.16	0.05	0.09	
<b>RENO</b>	1.61	1.96	1.26	0.12	0.04	0.08	
<b>HYBLA</b>	1.67	2.10	1.24	0.17	0.06	0.10	
	<b>SERVER</b>	<b>WESTWOOD</b>	<b>DELAY</b>	20ms			
	<b>Average(s)</b>	<b>Max(s)</b>	<b>Min(s)</b>	<b>Stdev(s)</b>	<b>95% CI(s)</b>	<b>COV</b>	
<b>BIC</b>	2.71	3.30	1.97	0.30	0.10	0.11	
<b>CUBIC</b>	2.75	3.31	1.94	0.35	0.11	0.13	
<b>HIGHSPEED</b>	2.84	3.34	2.15	0.24	0.08	0.09	
<b>VENO</b>	2.76	3.23	2.00	0.29	0.09	0.11	
<b>VEGAS</b>	2.84	3.24	2.07	0.22	0.07	0.08	
<b>YEAH</b>	2.80	3.32	2.19	0.29	0.09	0.10	
<b>WESTWOOD</b>	2.76	3.29	2.06	0.33	0.11	0.12	
<b>LP</b>	2.83	3.41	2.03	0.34	0.11	0.12	
<b>SCALABLE</b>	2.79	3.16	2.07	0.25	0.08	0.09	
<b>RENO</b>	2.79	3.21	2.12	0.25	0.08	0.09	
<b>HYBLA</b>	2.72	3.33	2.07	0.27	0.09	0.10	

Table 7.15: Values for WESTWOOD vs other TCP congestion algorithms

	SERVER	LP	DELAY	5ms		
	Average(s)	Max(s)	Min(s)	Stdev(s)	CI	COV
BIC	1.16	2.35	0.81	0.38	0.12	0.33
CUBIC	1.09	1.63	0.87	0.22	0.07	0.20
HIGHSPEED	1.08	2.26	0.79	0.29	0.09	0.27
VENO	1.03	1.66	0.85	0.16	0.05	0.16
VEGAS	1.06	1.53	0.89	0.15	0.05	0.14
YEAH	1.11	2.45	0.83	0.33	0.11	0.30
WESTWOOD	1.17	2.77	0.83	0.45	0.15	0.39
LP	1.13	3.71	0.87	0.47	0.15	0.41
RENO	1.13	2.36	0.87	0.35	0.11	0.31
SCALABLE	1.12	1.98	0.85	0.25	0.08	0.22
HYBLA	1.12	1.98	0.85	0.25	0.08	0.22
	SERVER	LP	DELAY	10ms		
	Average(s)	Max(s)	Min(s)	Stdev(s)	CI	COV
BIC	1.67	2.00	1.36	0.16	0.05	0.09
CUBIC	1.70	2.08	1.39	0.18	0.06	0.11
HIGHSPEED	1.67	2.02	1.41	0.13	0.04	0.08
VENO	1.74	2.47	1.24	0.23	0.07	0.13
VEGAS	1.69	1.98	1.28	0.14	0.04	0.08
YEAH	1.69	2.24	1.24	0.16	0.05	0.09
WESTWOOD	1.62	1.98	1.34	0.14	0.04	0.08
LP	1.64	2.04	1.21	0.18	0.06	0.11
RENO	1.67	2.04	1.37	0.15	0.05	0.09
SCALABLE	1.64	2.20	1.35	0.16	0.05	0.10
HYBLA	1.71	2.34	1.16	0.22	0.07	0.13
	SERVER	LP	DELAY	20ms		
	Average(s)	Max(s)	Min(s)	Stdev(s)	CI	COV
BIC	3.03	5.44	2.10	0.65	0.21	0.22
CUBIC	3.08	5.78	2.06	0.65	0.21	0.21
HIGHSPEED	3.02	5.71	2.04	0.58	0.19	0.19
VENO	3.04	5.57	2.31	0.51	0.16	0.17
VEGAS	3.14	5.66	2.02	0.72	0.23	0.23
YEAH	3.02	5.25	2.36	0.58	0.18	0.19
WESTWOOD	3.10	5.53	2.20	0.73	0.23	0.24
LP	3.12	6.79	2.26	0.80	0.26	0.26
RENO	3.27	6.17	2.31	0.84	0.27	0.26
SCALABLE	2.90	3.49	2.19	0.30	0.10	0.10
HYBLA	3.10	5.96	2.52	0.55	0.18	0.18

Table 7.16: Values for LP vs other TCP congestion algorithms

	SERVER	SCALABLE	DELAY	5ms		
	Average(s)	Max(s)	Min(s)	Stdev(s)	95% CI(s)	COV
BIC	0.63	1.04	0.57	0.09	0.03	0.14
CUBIC	0.73	1.29	0.57	0.19	0.06	0.27
HIGHSPEED	0.65	0.77	0.56	0.07	0.02	0.10
VENO	0.62	0.69	0.58	0.04	0.01	0.06
VEGAS	0.63	1.02	0.56	0.11	0.04	0.18
YEAH	0.66	1.01	0.58	0.10	0.03	0.15
WESTWOOD	0.62	1.03	0.56	0.08	0.03	0.13
LP	0.61	0.75	0.56	0.04	0.01	0.07
RENO	0.64	0.93	0.55	0.08	0.03	0.12
SCALABLE	0.64	0.93	0.55	0.08	0.03	0.12
HYBLA	0.61	0.84	0.56	0.06	0.02	0.09
	SERVER	SCALABLE	DELAY	10ms		
	Average(s)	Max(s)	Min(s)	Stdev(s)	95% CI(s)	COV
BIC	0.93	1.00	0.89	0.03	0.01	0.03
CUBIC	0.95	1.09	0.90	0.04	0.01	0.04
HIGHSPEED	1.04	1.80	0.88	0.23	0.07	0.22
VENO	0.94	1.09	0.87	0.05	0.01	0.05
VEGAS	0.94	0.99	0.90	0.02	0.01	0.02
YEAH	0.94	1.09	0.86	0.05	0.01	0.05
WESTWOOD	0.96	1.20	0.89	0.05	0.02	0.05
LP	0.97	1.20	0.88	0.09	0.03	0.09
RENO	1.02	1.61	0.90	0.16	0.05	0.15
SCALABLE	0.95	1.18	0.88	0.06	0.02	0.06
HYBLA	0.94	1.17	0.86	0.07	0.02	0.07
	SERVER	SCALABLE	DELAY	20ms		
	Average(s)	Max(s)	Min(s)	Stdev(s)	95% CI(s)	COV
BIC	1.70	2.64	1.48	0.20	0.06	0.12
CUBIC	1.60	1.94	1.38	0.09	0.03	0.06
HIGHSPEED	1.58	1.77	1.42	0.07	0.02	0.04
VENO	1.70	2.45	1.51	0.18	0.06	0.11
VEGAS	1.57	1.97	1.39	0.10	0.03	0.07
YEAH	1.64	2.00	1.43	0.12	0.04	0.07
WESTWOOD	1.76	2.18	1.53	0.16	0.05	0.09
LP	1.68	2.31	1.51	0.16	0.05	0.10
RENO	1.67	2.05	1.48	0.11	0.04	0.07
SCALABLE	1.63	2.39	1.49	0.15	0.05	0.09
HYBLA	1.72	2.45	1.54	0.18	0.06	0.11

Table 7.17: Values for SCALABLE vs other TCP congestion algorithms

	SERVER	RENO	DELAY	5ms		
	Average(s)	Max(s)	Min(s)	Stdev(s)	95% CI(s)	COV
BIC	0.94	1.20	0.59	0.13	0.04	0.14
CUBIC	0.99	1.27	0.73	0.12	0.04	0.12
HIGHSPEED	0.97	1.21	0.73	0.13	0.04	0.13
VENO	0.98	1.23	0.74	0.11	0.03	0.11
VEGAS	0.97	1.22	0.73	0.12	0.04	0.12
YEAH	0.96	1.22	0.74	0.11	0.03	0.11
WESTWOOD	0.96	1.24	0.77	0.09	0.03	0.10
LP	0.98	1.18	0.76	0.10	0.03	0.10
RENO	0.98	1.41	0.74	0.14	0.04	0.14
SCALABLE	1.00	1.18	0.83	0.11	0.03	0.11
HYBLA	0.96	1.18	0.75	0.08	0.02	0.08
	SERVER	RENO	DELAY	10ms		
	Average(s)	Max(s)	Min(s)	Stdev(s)	95% CI(s)	COV
BIC	1.69	1.93	1.22	0.16	0.05	0.09
CUBIC	1.68	2.16	1.32	0.15	0.05	0.09
HIGHSPEED	1.65	1.99	1.24	0.16	0.05	0.10
VENO	1.66	2.01	1.24	0.19	0.06	0.11
VEGAS	1.69	2.10	1.26	0.19	0.06	0.11
YEAH	1.66	2.04	1.24	0.18	0.05	0.11
WESTWOOD	1.67	2.00	1.24	0.15	0.05	0.09
LP	1.65	1.98	1.24	0.17	0.05	0.10
RENO	1.63	2.01	1.22	0.18	0.06	0.11
SCALABLE	1.68	2.01	1.39	0.16	0.05	0.09
HYBLA	1.59	1.84	1.24	0.09	0.03	0.06
	SERVER	RENO	DELAY	20ms		
	Average(s)	Max(s)	Min(s)	Stdev(s)	95% CI(s)	COV
BIC	2.73	3.26	2.09	0.25	0.08	0.09
CUBIC	2.63	3.22	2.10	0.30	0.09	0.11
HIGHSPEED	2.69	3.29	2.15	0.23	0.07	0.08
VENO	2.69	3.12	2.09	0.24	0.07	0.09
VEGAS	2.81	3.18	2.15	0.22	0.07	0.08
YEAH	2.73	3.22	2.14	0.30	0.09	0.11
WESTWOOD	2.72	3.32	2.01	0.32	0.10	0.12
LP	2.77	3.10	2.06	0.21	0.07	0.08
RENO	2.65	3.14	2.06	0.29	0.09	0.11
SCALABLE	2.71	3.26	1.96	0.29	0.09	0.11
HYBLA	2.72	3.29	2.13	0.23	0.07	0.08

Table 7.18: Values for RENO vs other TCP congestion algorithms

	SERVER	HYBLA	DELAY	5ms		
	Average(s)	Max(s)	Min(s)	Stdev(s)	95% CI(s)	COV
BIC	0.91	1.08	0.56	0.09	0.03	0.10
CUBIC	0.91	1.13	0.70	0.07	0.02	0.08
HIGHSPEED	0.91	1.20	0.74	0.07	0.02	0.08
VENO	0.92	1.13	0.79	0.06	0.02	0.06
VEGAS	0.90	1.02	0.73	0.06	0.02	0.07
YEAH	0.91	1.19	0.74	0.08	0.03	0.09
WESTWOOD	0.92	1.13	0.74	0.06	0.02	0.07
LP	0.91	1.21	0.74	0.09	0.03	0.10
RENO	0.91	1.12	0.73	0.07	0.02	0.08
SCALABLE	0.93	1.14	0.74	0.07	0.02	0.07
HYBLA	0.91	1.12	0.73	0.06	0.02	0.07
	SERVER	HYBLA	DELAY	10ms		
	Average(s)	Max(s)	Min(s)	Stdev(s)	95% CI(s)	COV
BIC	1.60	1.81	1.22	0.11	0.03	0.07
CUBIC	1.55	2.01	1.14	0.16	0.05	0.10
HIGHSPEED	1.56	1.74	1.24	0.11	0.03	0.07
VENO	1.58	1.79	1.24	0.10	0.03	0.06
VEGAS	1.55	1.76	1.24	0.12	0.04	0.08
YEAH	1.58	1.94	1.41	0.10	0.03	0.06
WESTWOOD	1.58	1.82	1.24	0.10	0.03	0.06
LP	1.58	1.92	1.17	0.13	0.04	0.08
RENO	1.55	1.74	1.22	0.08	0.03	0.05
SCALABLE	1.57	1.96	1.22	0.12	0.04	0.07
HYBLA	1.55	1.76	1.22	0.11	0.03	0.07
	SERVER	HYBLA	DELAY	20ms		
		Max(s)	Min(s)	Stdev(s)	95% CI(s)	COV
BIC	2.25	3.02	1.69	0.33	0.10	0.15
CUBIC	2.21	3.02	1.67	0.43	0.13	0.19
HIGHSPEED	2.33	2.84	1.74	0.34	0.11	0.15
VENO	2.29	2.85	1.73	0.33	0.10	0.15
VEGAS	2.25	3.07	1.62	0.32	0.10	0.14
YEAH	2.23	2.92	1.66	0.39	0.12	0.18
WESTWOOD	2.36	2.87	1.62	0.32	0.10	0.14
LP	2.40	2.99	1.70	0.42	0.13	0.17
RENO	2.25	2.89	1.65	0.39	0.12	0.17
SCALABLE	2.19	2.96	1.69	0.43	0.13	0.20
HYBLA	2.30	3.00	1.61	0.40	0.12	0.17

Table 7.19: HYBLA vs other TCP congestion algorithms

## 7.5 Graphs for Scenario B

The following are the graphs obtained from different scenarios after the experimentation has been done. These graphs have been plotted using Microsoft excel.

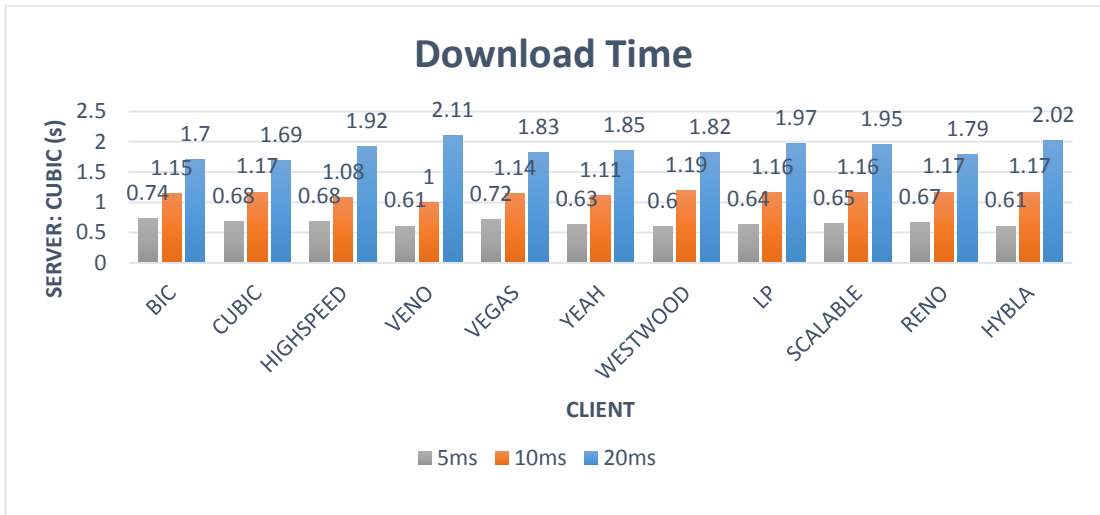


Figure 4: Download time for TCP-CUBIC on server side by varying delays

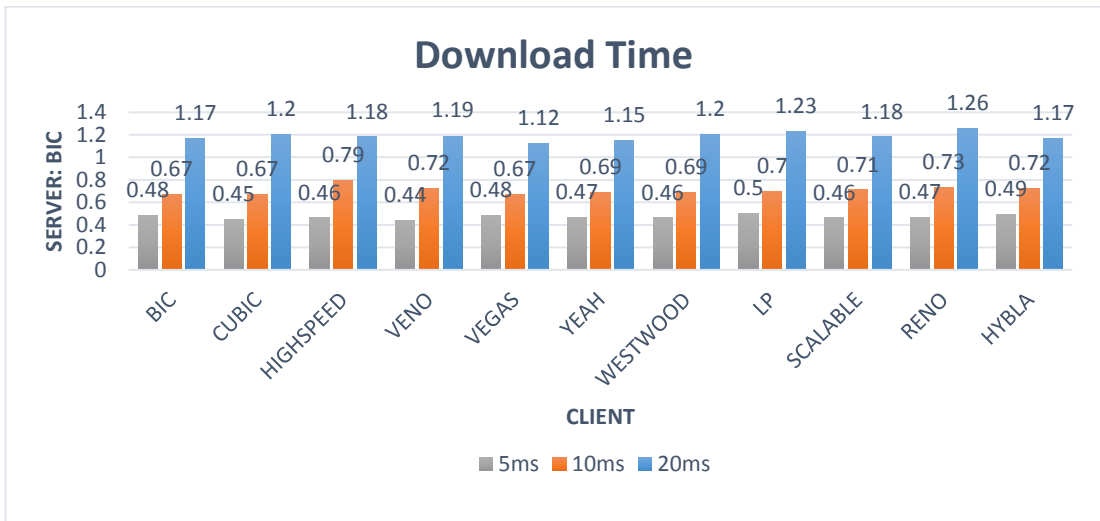


Figure 5: Download time for TCP-BIC on server side by varying delays

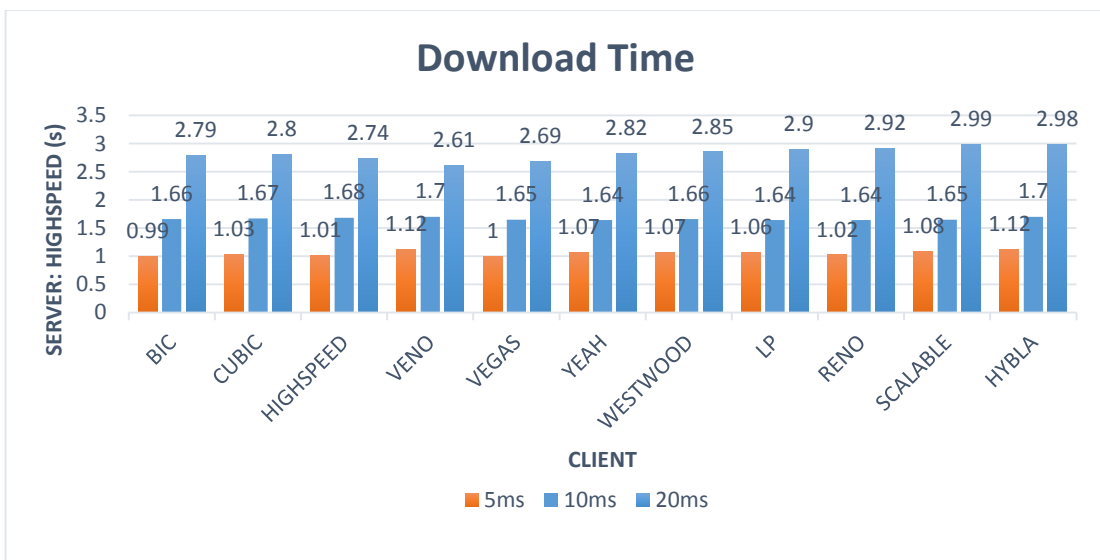


Figure 6: Download time for TCP-HIGHSPEED on server side by varying delays

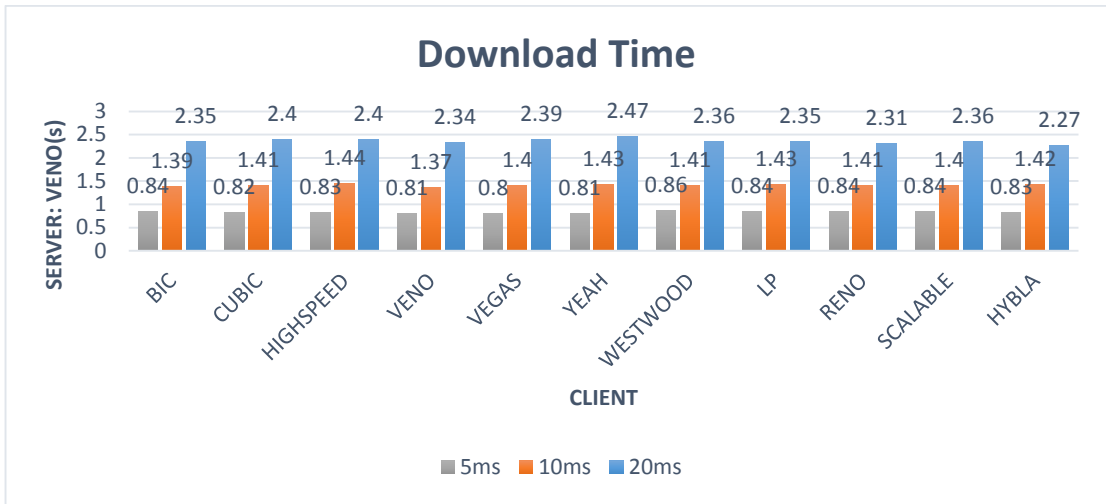


Figure 7: Download time for TCP-VEÑO on server side by varying delays

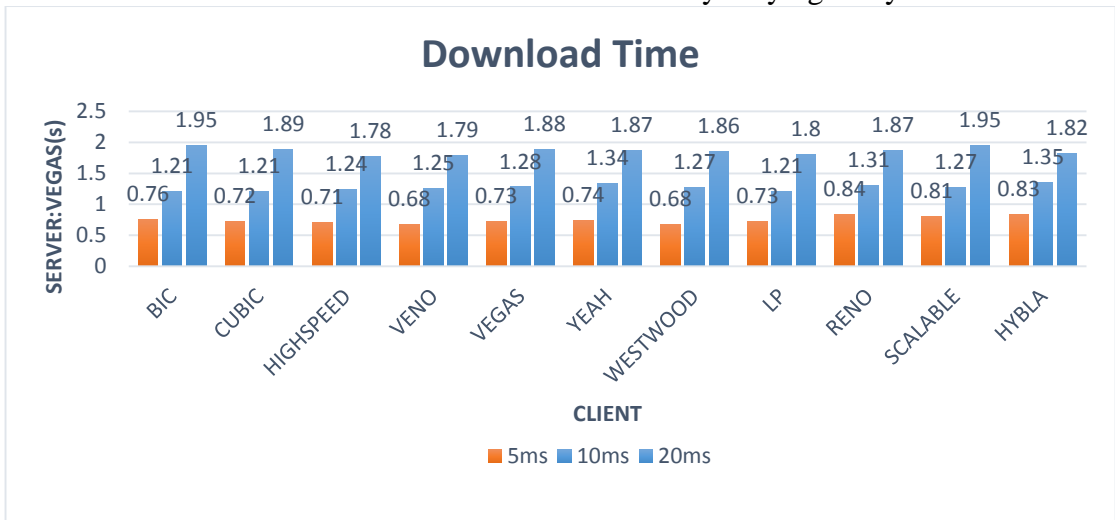


Figure 8: Download time for TCP-VEGAS on server side by varying delays

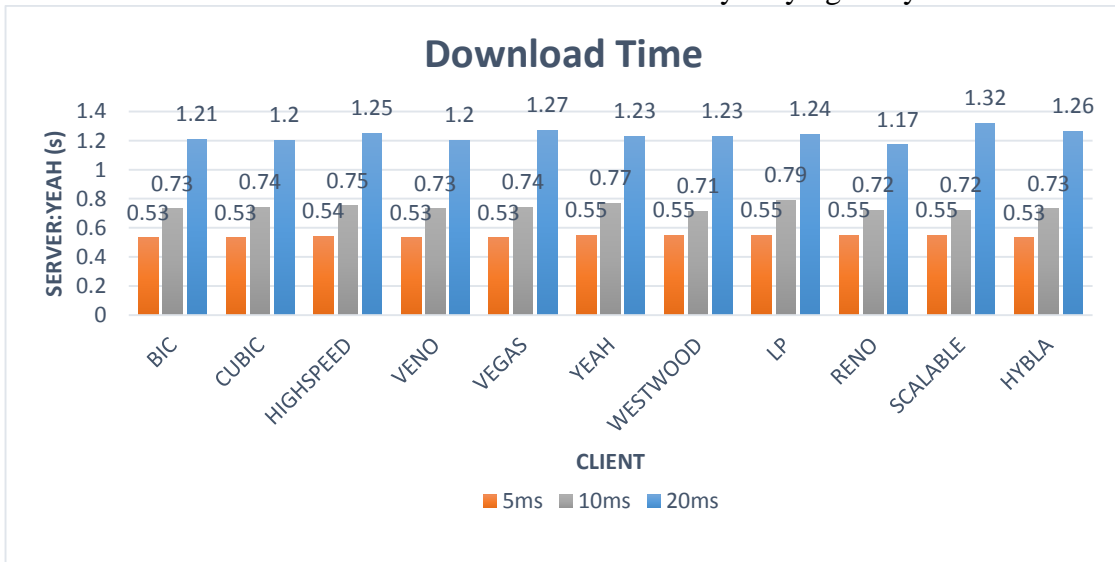


Figure 9: Download time for TCP-YEAH on server side by varying delays

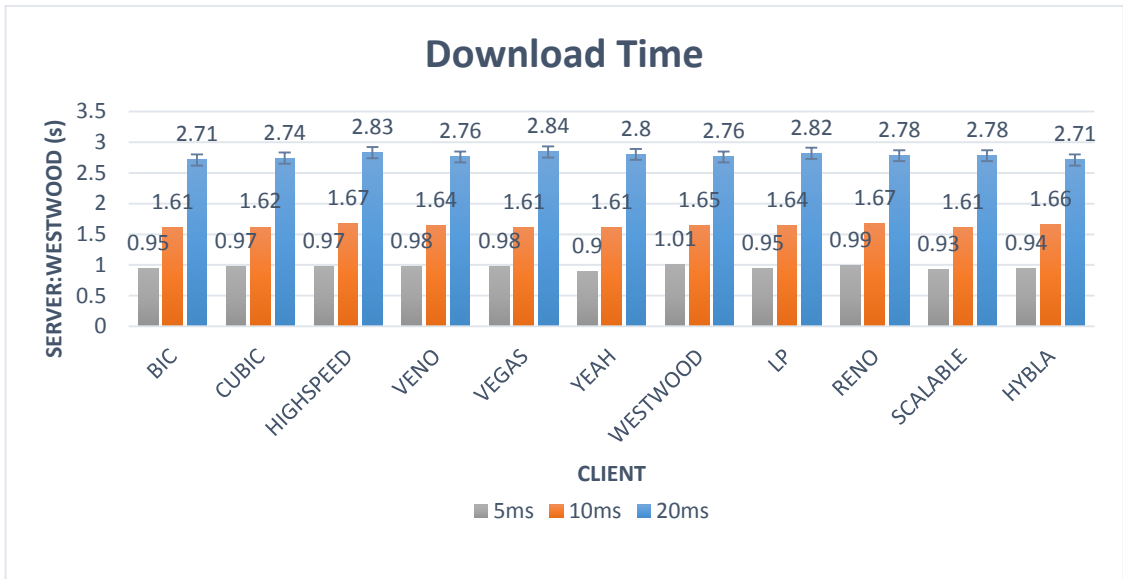


Figure 10: Download time for TCP-WESTWOOD on server side by varying delays

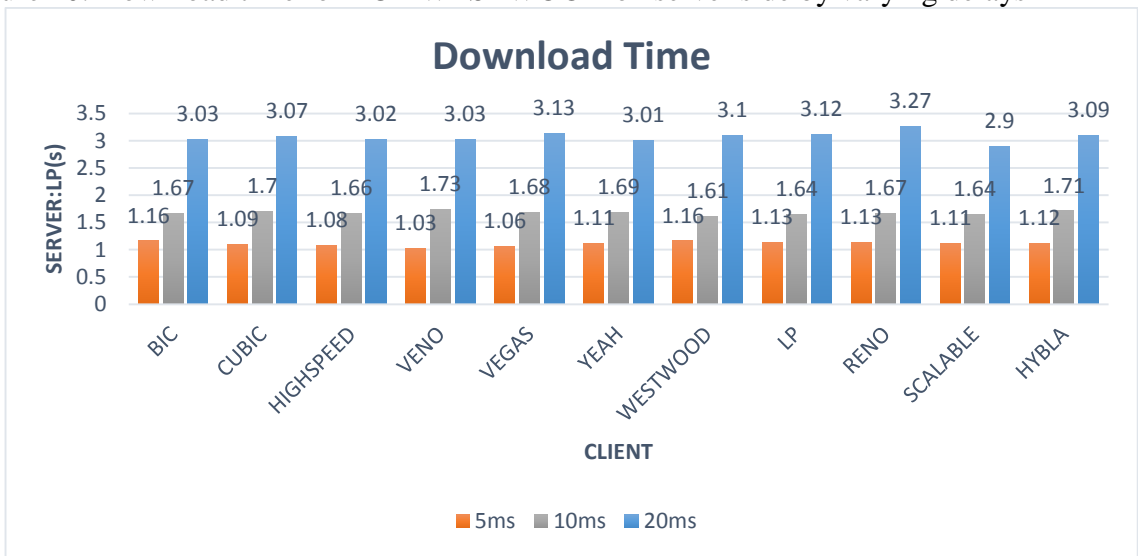


Figure 11: Download time for TCP-LP on server side by varying delays

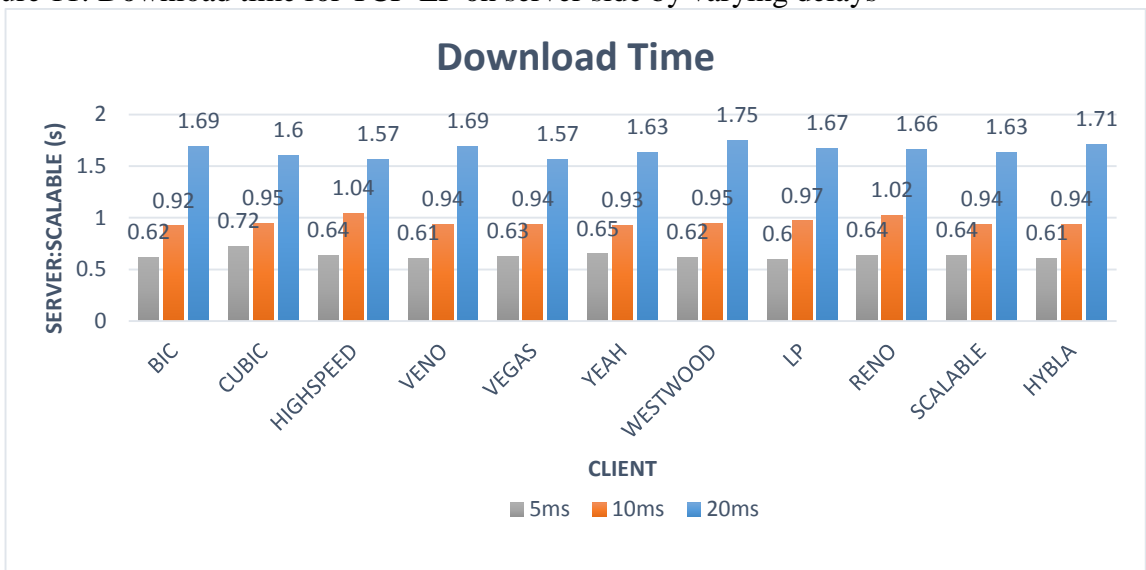


Figure 12: Download time for TCP-SCALABLE on server side by varying delays

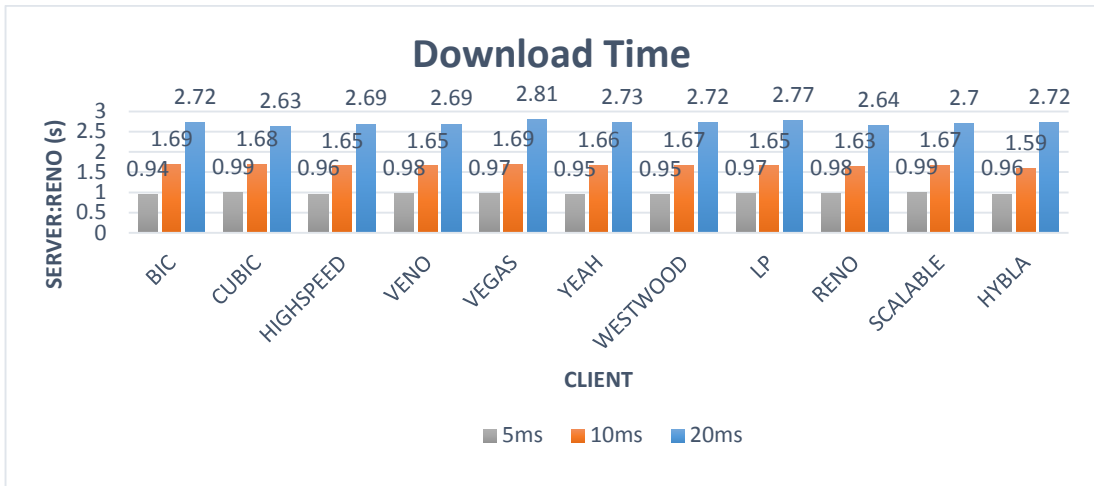


Figure 13: Download time for TCP-RENO on server side by varying delays

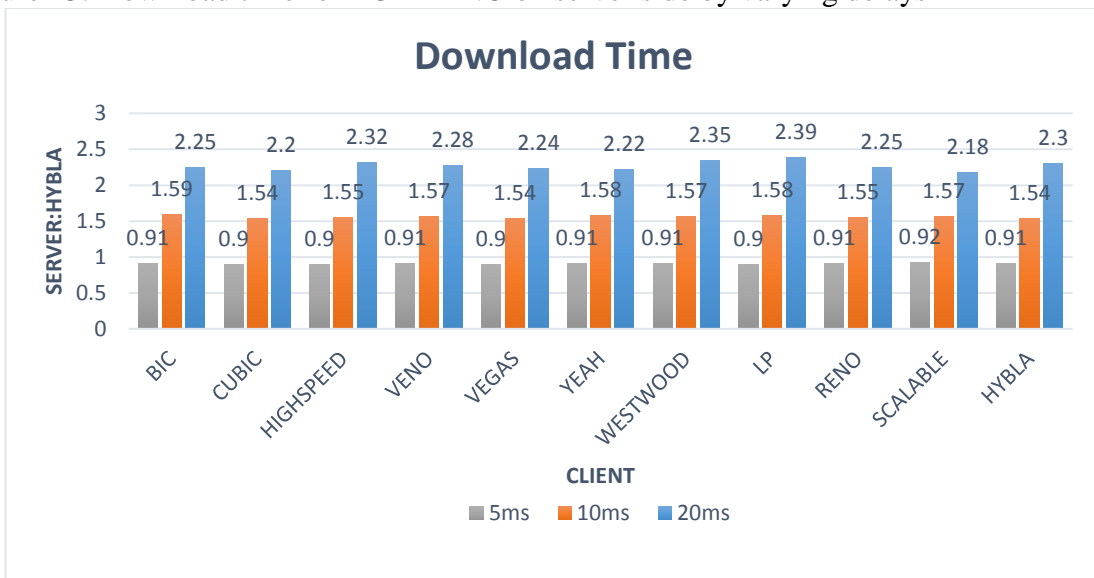


Figure 14: Download time for TCP-HYBLA on server side by varying delays