Analysis of TCP Variants over Variable Traffic

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Technology

in

Computer Science and Applications

Submitted By

Varun Chauhan

(601303029)

Under the supervision of:

Dr. Rajesh Kumar

Associate Professor

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

THAPAR UNIVERSITY

PATIALA – 147004

June 2015

Certificate

I hereby certify that the work which is being presented in the thesis entitled, "*Analysis of TCP Variants over Variable Traffic*", in partial fulfillment of the requirements for the award of degree of Master of Technology in *Computer Science and Application* submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of *Dr. Rajesh Kumar* and refers other researcher's work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

(Varun Chauhan)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

(Dr. Rajesh Kumar)

Associate Professor,
CSED

Countersigned by

(Dr. Deepak Garg)

Head

Computer Science and Engineering Department

Thapar University

Patiala

(Dr. S. S. Bhatia)

Dean (Academic Affairs)

Thapar University

Patiala

Acknowledgement

I would like to express my sincere gratitude to my mentor and supervisor Dr. Rajesh Kumar for his immense help, guidance, stimulating suggestion and full time encouragement. He always provided a motivational and enthusiastic atmosphere to work with, it was a great pleasure to do dissertation under his supervision.

I am also thankful to Dr. Deepak Garg, Head, Computer Science & Engineering Department, Ms. Sanmeet Kaur, C.S.A. Coordinator and Mr. Vishal Sharma, Research scholar for their constant support and encouragement.

I would like to thank all the faculty members and staff of the department who were always there at the need of the hour and provided with all the help and facilities, which I required for the completion of this work.

I express my thanks to my family for their love, support and enthusiastic encouragement without which I could not complete this dissertation. I also want to thank my friends for optimism and moral support throughout the completion of this work. Finally I thank the Almighty who gave me the strength to complete this work.

(Varun Chauhan)

601303029

Abstract

In today's scenario most of the work carried out is accomplished through the internet. Internet plays a vital role in day-to-day's regular activities and hence communication has evolved to a new level. But reliability and efficiency of data transfer on the Internet is an important issue. Since late 70's, the protocol responsible for that has been the de facto standard TCP, which has proven to be successful throughout the years, its self-managed congestion control algorithms have retained the stability of the Internet for decades. It has emerged as the most dominant end to end protocol in this era of internet and has been foreseen to remain the same in future.

However, the variety of existing new technologies such as high-speed networks (e.g. fibre optics) with high-speed long-delay set-up (e.g. cross-Atlantic links) and wireless technologies have posed lots of challenges to TCP congestion control algorithms. Due to the unawareness of the network condition, regular TCP is unable to handle resources of the network as well as cannot differentiate between congestion loss and random loss of packet. So the congestion control research community worked upon the issues and proposed solutions to most of these challenges. TCP has got many enhanced version for wired, wireless and high speed networks to improve the trait of basic TCP such as Tahoe, Reno, New-Reno, SACK and so on. Each variant performs in a different routine for different networks based upon the various key parameters. These days presence of TCP can be easily spotted among various applications.

In this paper we have simulated various TCP variants with respect to different network parameters such as variable packet size, bandwidth and buffer size. This paper analyze the throughput, packet drop, delay and jitter of TCP variants evaluated over the above mentioned network parameters.

Table of Contents

Chapter 1
Introduction

1.1 Transmission Control Protocol (TCP)

TCP stands for Transmission Control Protocol. TCP is a reliable end-to-end protocol which adapts to network conditions to guarantee transmission of data. It was designed to provide reliable performance even when the network on which it runs is unreliable.

It is the Transmission Control Protocol, rather than the Internet protocol (IP), which is responsible for guaranteeing the delivery of packets. This means that TCP needs to implement timers or some other mechanism to discern when packet loss occurs. It also needs to retransmit packets when they are lost. TCP uses a sliding window protocol to determine when packet loss has occurred. This works as follows:

- Whenever the sender transmits a segment, it also starts a timer
- When the segment arrives at the receiver, the receiver sends back a segment (which may or may not contain data, depending on whether the receiver has any data it wishes to transmit) which include an acknowledgement number which identifies the next segment the receiver expects to receive.
- If the sender's timer expires before it receives the acknowledgement (ACK), then the sender retransmits the segment.
- Otherwise (i.e. if the ACK is received before the timer expires) the segment has been received successfully and no further action is necessary.

This protocol is not foolproof. It is possible for segments to experience long delays in transmission. They eventually arrive at the receiver, but in the meantime the sender has timed out and retransmitted such packets since it believes that those have been lost. It is also possible that segments arrive at the receiver out of order (it is the TCP receiver which is responsible for rearranging packets into the correct order once they have been received) causing some packets to be received but not acknowledged since TCP acknowledges segments in order. Hence higher numbered packets may be received but

cannot be acknowledged until a lower numbered segment, which has taken a different route through the network and been delayed, arrives. If this occurs, it is possible that the sender will time out for these higher numbered segments since the delay in acknowledgement is so high. The sender will then retransmit all such segments, even though this is unnecessary.

Another function that TCP offers is the correct reassembly of segments arriving out of order. TCP transmits segments as IP datagrams, and since IP datagrams can arrive out of order, the receiver side of the connections has to re-sequence the data and pass them in the correct order to the application.

A TCP segment consists of header and data part. The header maintains all the information that is needed in order for TCP to be able to provide all the functions discussed before. Its normal size is 20 bytes except if options are present. In that case, the length is determined by the data offset, options can have a variable length of 0-320 bits (should be divisible by 32, padding is used otherwise). The value of the data field depends on the maximum segment size (MSS) of the path and the length of the header.
Figure 1.1 shows the structure of a TCP segment. It has following fields:

- **16 bit source port address**: This field identifies the port from which the segment was sent.
- **16 bit destination address:** This field identifies the port to which this segment is to be sent.
- **32 bit sequence number:** This field has a dual role. If the SYN flag is set, It will be the initial sequence number. The sequence number of the actual first data byte and the acknowledged number in the corresponding ACK is then equivalent to sequence number plus 1. If the SYN flag is not set, then this is the cumulative sequence number of the first data byte of the segment for this connection.
- **32-bit acknowledgment number**: If the ACK flag is set, then this is the next sequence number that the receiver is expecting. This field acknowledges the sequence number up to this value

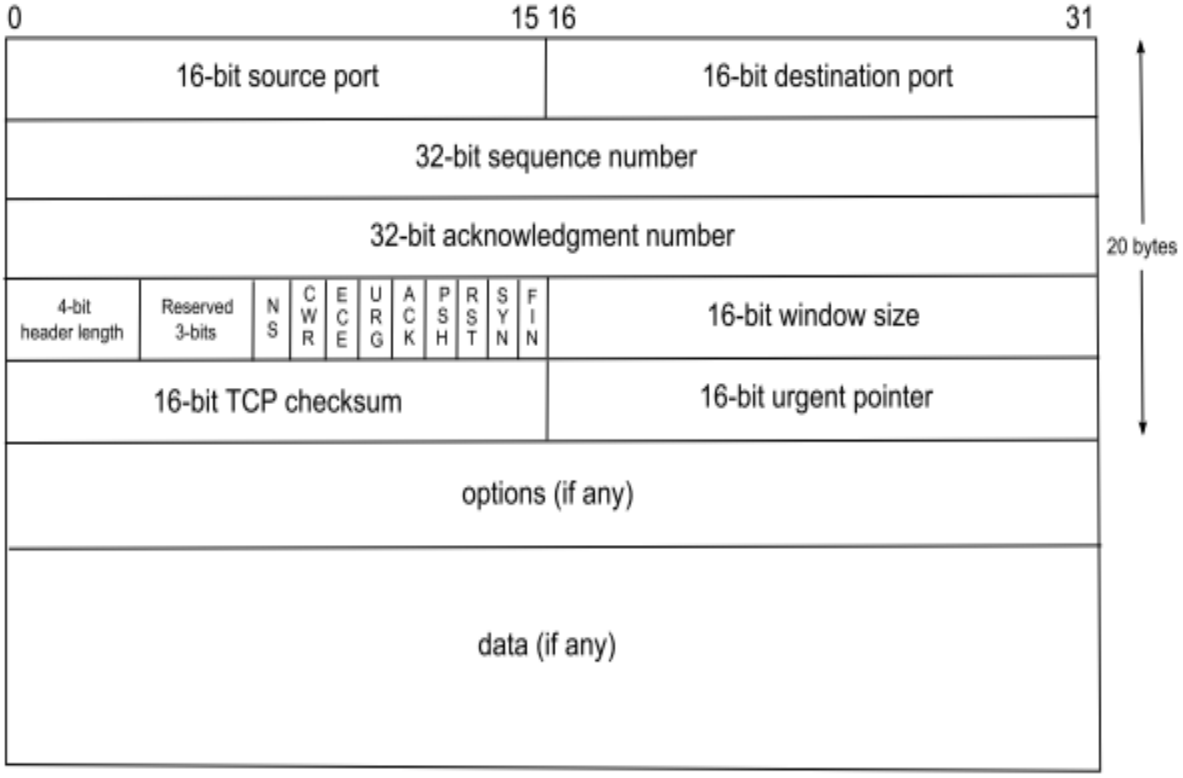


Figure 1.1: TCP Header

- **4-bit header length:** This field specifies the size of the TCP header in 32-bit words. The minimum size of header is 20 bytes and maximum of 60 bytes thus giving us minimum header size of 5 words and the maximum of 15 words. This allows us to make use of 40 bytes of options in the header.
- **Reserved bits:** This field is reserved for future use and should be set to 0.
- **Flags:** There are nine control flags used in a TCP segment. SYN, ACK and URG, when set, this indicate that the value in the corresponding field is valid. NS, CWR and ECE are used for Explicit Congestion Notification (ECN). FIN is used to indicate the end of a connection. PSH is used to perform a push action. It requests to push the buffered data to the application. Lastly, RST is used to reset a connection.
- **16-bit window size:** The size of the receive window in bytes that the sender of this segment is currently willing to receive. In most cases, it is the current size of the buffer of the receiving application.

- **16-bit TCP checksum**: The checksum field that is used for error checking.
- **16-bit urgent pointer**: If the URG flag is set, this field indicates the offset from the sequence number pointing the last urgent data byte.
- **Options**: This field is optional and carries TCP options such as the MSS, the amount and sequence numbers of Selective Acknowledgment (SACK) blocks etc. MSS is used to indicate the maximum segment size allowed in the path and SACK is used by a receiver to notify a sender for delivered off-sequence segments.
- **Data**: This optional field carries the actual data that an application wants to transmit.

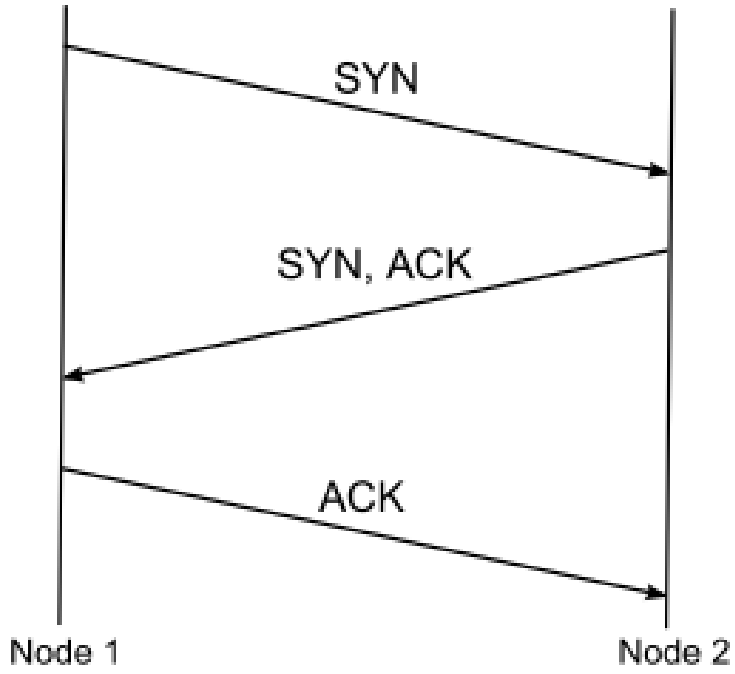


Figure 1.2: TCP Initial Handshake

As already mentioned, TCP is a connection oriented protocol. This means that in order to reliably transfer a data stream from one end of a network to another, it first needs to establish a logical connection using a three-way handshake. Figure 2.2 shows this procedure. Node 1 sends a segment with the SYN flag set to Node 2, in order to initiate a connection. Node 2 replies with a SYN-ACK, meaning that both of these flags are set. Node 1 acknowledges then that a connection is now active with an ACK. The connection

is now set and the two nodes can exchange data. Note that these first segments can also carry information useful for the initialization of the connection such as the MSS, the initial sequence number etc. A recent extension proposes that a connection can carry data while performing the three-way handshake. This approach can significantly speed up connections that require only a few round trips, something that is common in today's internet.

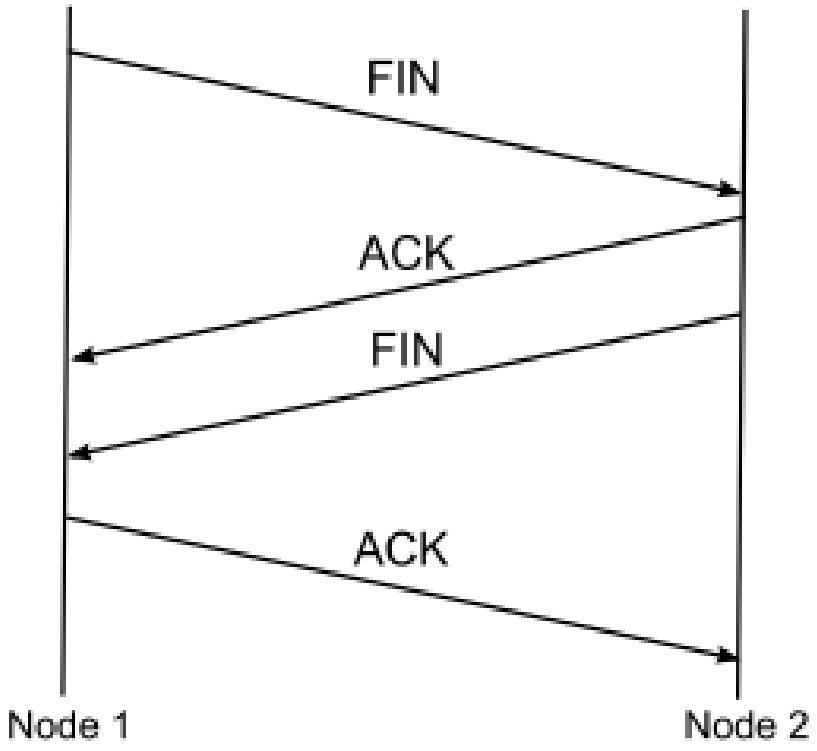


Figure 1.3: Termination of TCP Connection

While it takes three segments to establish a connection, it takes four to terminate the connection. This is caused by TCP's half-close. Since each TCP connection is full duplex, meaning that data can flow to each direction independently, each part of the connection has to close independently. The rule is that both ends can send a FIN when it has no more data to send. The receipt of the FIN means that there will not be any other data flowing in that direction. Figure 1.3 shows the termination of a TCP connection. Node 1 finishes sending data and thus sends a FIN. Node 2 acknowledges the FIN and sends its own FIN to Node 1. Lastly, Node 1 acknowledges the FIN from Node 2. The

connection is now closed. Note that for both the initiation and termination of a connection, the simplest scenarios are presented. There can be many issues, e.g., a FIN or ACK might get lost. Since these are normal segments the usual procedure described in the next section will be followed in order to recover the losses.

The size of the segments which are transmitted is decided by the TCP software. However, each segment including its header, must fit into 65,515 bytes, which is the size of the IP payload. The size of segments may also be constrained by the segment size of the network over which they are being transmitted and by the amount of data the receiver can cope up. These considerations aside, TCP may put data from more than one write operation into one segment. Similarly, it may split the data from one write and put it into several separate packets. Irrespective of segment size, every TCP segment has a header which is always 20 bytes. It may also include data, although it is permissible for a segment to be solely a header.

When an application passes data to TCP to be transmitted, the TCP software may decide to transmit it immediately, or it may buffer the data. The application can override this decision to buffer in the case of urgent data, by using a PUSH flag.

A TCP connection requires both the sender and the receiver to create TCP sockets and to explicitly establish a connection between these sockets. This connection is full duplex i.e. data can travel in both directions at the same time. TCP uses acknowledgements and for this reason does not support multicasting or broadcasting since the large number of acknowledgement packets which would be generated in such circumstances could overwhelm the sender.

1.2 Flow Control

In TCP, flow control ensures that a sender does not send more data than a receiver can handle. Since the data arrival at the receiver must be in exactly the same order that was sent by the application, a TCP receiver should buffer out-of-order data until all gaps in the stream are filled before forwarding the complete and ordered stream to the receiving application. The receiver specifies a receiver window (rwnd) in every ACK that it sends

back to the sender to inform the sender that how many bytes are allowed to send without overloading the receiver buffer.

1.3 Congestion Control

In 1986, the Internet (ARPANET) faced its first congestion collapse. The end hosts transmitted as much data as they could, where the routers were not able to handle them, Due to the retransmission of the lost segments, the transmission rate did not get lower either. Van Jacobson proposed a mechanism called congestion control which has been implemented in TCP. Congestion control is TCP's attempt of not sending more data than what can be in flight from sender to receiver.

The two values that limit the number of bytes a sender is allowed to transmit before receiving any ACK back are: the receiver window (rwnd) and the congestion window (cwnd). The first is used for flow control and is dictated by receiver, while the second is used to avoid network congestion and is an estimated value defined by the sender. The congestion window shows the estimated number of bytes that can be injected into the network without causing network congestion. A TCP sender must oblige both the rwnd and cwnd limits and calculates its sender window (swnd) as:

swnd = min(rwnd, cwnd)

In TCP congestion control, the congestion window is gradually increased until packet loss is detected. By the detection of packet loss, the congestion window is quickly reduced in order to avoid congestion, this behavior is called additive increase/multiplicative decrease (AIMD).

1.3.1 Slow Start and Congestion Avoidance

There are two phases called slow start and congestion avoidance which they control the congestion window growth. In slow start phase, the capacity of the network is estimated by quickly increasing the number of packet sent per round trip time (RTT). This phase ends as soon as the cwnd reaches the threshold or packet loss is detected, which shows that the data rate is overloading the path capacity. In congestion avoidance phase, the cwnd is increased less quickly than slow start phase. The border between these

two phases is slow start threshold (sshthresh) which can be referred as a value for sending data without congestion. When TCP establishes connection for the first time, it starts in the slow start phase, with the congestion window size of 1 segment and the slow start threshold set to a randomly high value. During slow start, the congestion window is increased by 1 Maximum Segment Size (MSS) for each incoming ACK. During congestion avoidance, the congestion window is increased by 1 MSS per RTT.

The cwnd growth will stop when the sender detect that a segment is missing at the receiver, whether by receiving DupACK or if RTO expires and timeout occurs. In both cases, standard TCP interprets this loss of data as sign of network congestion and reduces both cwnd and ssthresh.

As discussed in previous section, a TCP sender tries to recover from segment loss either after RTO expiration or after receiving 3 DupACK (fast retransmit). In both cases, the sshtresh will be reduced to half of the cwnd value.

ssthresh = cwnd/2

The difference is that, after timeout (i.e., RTO expiration), TCP will be forced back to slow start phase, in which the cwnd will be reset to 1. While after a fast retransmit, TCP goes to congestion avoidance phase by setting the cwnd equal to sshthresh.The standard TCP congestion control which is only based on RTO loss recovery and fast retransmit, normally referred as TCP Tahoe.

TCP sender utilizes slow start and congestion avoidance algorithm to determine how much data can be sent via current window. These algorithms use a variable called Congestion Window (cwnd) which gives the information to the sender about the amount of data that can be send over the network till an acknowledgement is received. Whether slow start algorithm or congestion avoidance algorithm should be used is decided by the value of Slow Start Threshold (ssthresh). The decision criteria are:

- When cwnd < ssthresh, then slow start algorithm is preferred over congestion avoidance algorithm.

- When cwnd>ssthresh, then congestion avoidance algorithm is preferred over slow start algorithm.

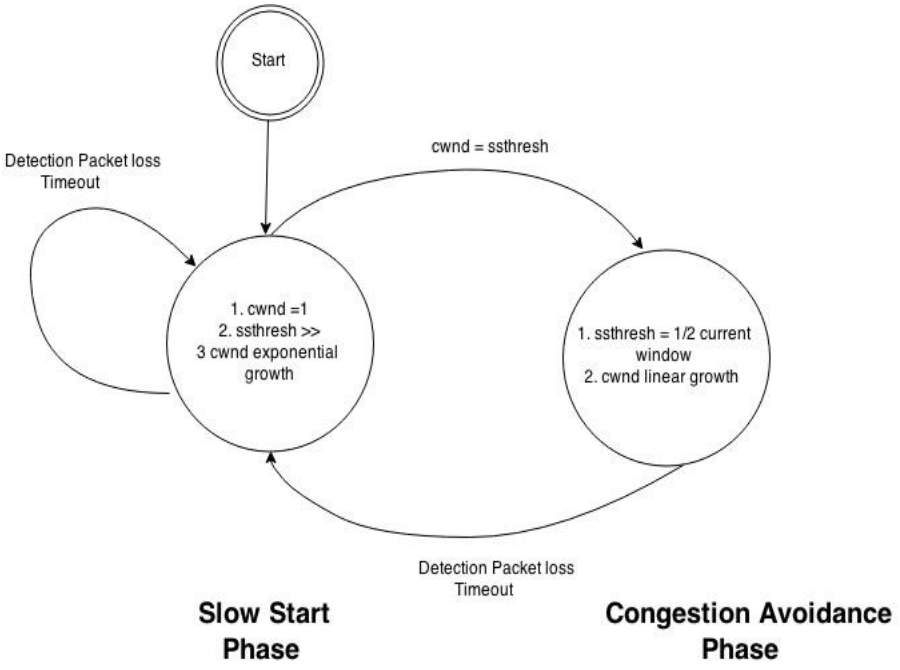


Figure 1.4: TCP FLOW DIAGRAM

1.3.2 Loss Recovery

When a segment is transmitted, TCP initiates a retransmission timer. The segment will be retransmitted, if the timer expires before the segment is acknowledged. This timer is called Retransmission Timeout (RTO) and its value is calculated dynamically based on the measurement of the time it takes from the transmission of a segment until the acknowledgement is received, referred as Round Trip Time (RTT). When a timeout occurs, the sender retransmits the lost segment and doubles the RTO (exponential backoff).

By this method, A TCP sender can sample the RTT upon receiving each ACK. But if the sender receives an ACK which contains a sequence number of a segment which had already been retransmitted, the sender cannot distinguish whether the ACK is the receiver's answer to the original segment (e.g. reordered) or to the retransmitted segment.

To overcome this problem use timestamp option field is used in TCP header. If both endpoints support this feature, then the current time will be included in timestamp option field in TCP header upon sending each segment. Therefore the correct and reliable RTT can be estimated. i.e. sender can distinguish whether the ACK from receiver is for the retransmitted segment or for the original segment which has been sent for the first time.

1.3.3 Fast Retransmit

In order to avoid long waiting time (i.e. one RTT) till the RTO expires for a missing segment, TCP employs a scheme called fast retransmit. It is possible that TCP receiver has some gaps of outstanding data in the received stream due to the fact that IP packets can get lost, duplicated or even reordered in the network. In this case, the receiver cannot increase the ACK number, although it is receiving the new data. Therefore it sends an unchanged ACK number upon arrival of each new data. This ACK which its number is identical to previous ACK is called a duplicate ACK (DupACK). This DupACK informs the sender that the receiver is missing one segment.

Fast retransmit assumes 3 consecutive DupACKs as the sign of packet loss. In this condition, the TCP sender resets the transmission timer and triggers fast retransmission.

1.3.4 Fast Recovery

The new algorithm prevents the communication route (pipe) from getting empty after Fast Retransmit, thereby avoiding the need to Slow-Start it after a single packet loss. Fast Recovery operates by assuming each DupACK received represents a single packet which left the pipe. Thus, during Fast Recovery the TCP sender is able to make intelligent estimates of the amount of outstanding data. Fast Recovery is entered by a TCP sender after receiving an initial threshold of dup ACKs. The sender's usable window is calculated as below.

Min(awin, cwnd, ndup)

Where awin is the receiver's advertised window, cwin is the sender's congestion window, and ndup is maintained until the number of DupACKs reaches threshold i.e 3 duplicate ACKs, and thereafter tracks the number of duplicate ACKs. Thus, during Fast Recovery the sender "inflates" its window by the number of DupACKs it has received. According

to the observation that each DupACK indicates some packets has been removed from the network and is now cached at the receiver. After entering Fast Recovery and retransmitting a single packet, the sender effectively waits until half a window of DupACKs has been received, and then sends a new packet for each of the duplicate ACK. Upon receipt of an ACK for new data (called a “recovery ACK”), the sender exits Fast Recovery by setting ndup to 0.

Chapter 2

Literature Survey

2.1 Historical Perspective

TCP is a connection oriented transport protocol which guarantees a reliable end-to-end delivery of data packets to the application layer. The application data that is submitted to the TCP is divided into segments before transmission. In order to achieve reliability TCP uses Automatic Repeat reQuest (ARQ) mechanism based on acknowledgments. Each byte is numbered in TCP, therefore the number of the first byte in a segment is used as a sequence number in the TCP header. In addition, a receiver sends a cumulative acknowledgement (ACK) in response to the incoming segment, which signifies that number of segments can be acknowledged at the same time.

2.2 Types to TCP Variants

Many other congestion control algorithms have been proposed in order to achieve better utilization of the available path capacity. They differ from each other in terms of how to detect congestion. In addition to packet loss, which is a sign of network congestion, other factors can also interpret as a sign of this phenomenon. For instance, an increase in RTT could be the result from congested router buffer. Therefore these different proposed congestion controls deal with different signs of congestion. Another difference is how they deal with congestion in terms of adjusting cwnd and sshthresh if it is detected.

The reason behind different proposed congestion controls algorithms is the fact that they are designed to improve the performance of TCP in different path environments. Some have been proposed for lossy wireless environments, while other works best in high speed networks or over satellite links with high latency.

There are different TCP variants based upon the several ways of implementation of congestion control algorithm. These variants are described below:

2.2.1 TCP Tahoe: A congestion control algorithm, TCP Tahoe had been proposed by V. Jacobson based on a concept of "Conservation of Packets". It works on the model in which a connection is working at available bandwidth capacity and then no new packet is injected in the network till the previous packet gets acknowledged by the receiver. Major executions in Tahoe are:

a. Slow Start: Whenever a connection starts or restarts after a packet loss, TCP Tahoe uses slow start. Since the packet loss rate is high, initially if huge number of packets is sent, it can congest the network and so TCP Tahoe set initial cwnd=1 and then increases exponentially until a sign of congestion is captured by it.
b. Congestion Avoidance: TCP Tahoe uses the concept of Additive increase multiplicative decrease (AIMD). When congestion occurs Tahoe sets the ssthresh as half of the current window value and then sets cwnd to one. Afterwards the slow start procedure is started until cwnd reaches the threshold value [1].

2.2.2 TCP Reno: Another congestion control algorithm, TCP Reno had been proposed by V. Jacobson. This variant sustains slow start and coarse grain retransmit timer concepts of the TCP Tahoe but adds an intelligence to detect the packet loss in minimum time period. In this, an algorithm called "Fast Retransmit" was suggested, which considered the occurrence of three duplicate acknowledgements for a same segment as the sign of packet loss and retransmits that packet without waiting for the time out. Also it does not set cwnd as one, but it is set to half after the detection of packet loss [2].

2.2.3 TCP New-Reno: A Modified version of congestion control algorithm of TCP Reno had been proposed by S. Floyd et. al, named as TCP New-Reno. This algorithm detects multiple packet loss and is more efficient than TCP-Reno while dealing with multiple packets. Like Reno, it enters into the Fast Retransmit algorithm as soon as the acknowledgment duplicacy is detected but unlike Reno, it does not come out of the fast recovery phase until all the outstanding packets are acknowledged [3].

2.2.4 TCP SACK: A congestion control algorithm, TCP SACK had been proposed by M. Mathis et. al. TCP SACK (selective acknowledgement) is an extension of TCP Reno

and TCP New-Reno. It overcomes the drawbacks of TCP Reno as well as the TCP New-Reno for eg detection of multiple packet loss and retransmission of more than one packet in single RTT (round trip time). It laid emphasis on making the selective acknowledgement rather than cumulative acknowledgement on a segment. Every ACK has a block, which provides the information to sender regarding acknowledged segments and using this information sender retransmits the unacknowledged segments [4].

2.2.5 TCP Vegas: A congestion control algorithm, TCP Vegas had been proposed by L.S. Brakmo et. al. It is the modified version of TCP Reno. It is based upon the fact that proactive measures in the congestion detection are more efficient as compared to the reactive measures. It modified slow start algorithm to prevent congestion in the network. It does not merely depend upon the packet loss as the only sign of congestion occurrence, rather it helps in detection of congestion prior occurring of packet loss in network [5].

2.2.6 TCP Westwood: A congestion control algorithm, TCP Westwood had been proposed by C. Casetti et. al. It is a sender side modification of congestion window algorithm which enhances the performance of TCP's on the wired as well as wireless network. Its performance is insensitive for the random loss of packets unlike TCP-Reno which was equally sensitive for random loss and congestion loss as it failed in determining the difference between the two. TCP Westwood completely follows end to end transmission, in which sender side's bandwidth is constantly examined by keeping a track of rate of returning of ACK's. This rate is further used to determine the values of cwnd and ssthresh unlike TCP-Reno, which blindly halves the ssthresh [6].

2.2.7 TCP Veno: A congestion control algorithm, TCP Veno had been proposed by C. P. Fu. It is simple and effective for dealing with random packet loss. The main idea is to monitor network congestion level and then the same information is further used to determine whether it is congestion error or random bit error. It works on the similar idea of Vegas but instead of using the measures for adjusting the window size proactively, it uses the measure to detect whether the connection is in congestion or not [7].

2.2.8 TCP H-TCP: A congestion control algorithm, TCP H-TCP had been proposed by R. Shorten et. al. This TCP was designed for the high speed and long distance networks. It stated that the conventional TCP variants were not effective when the window size was large and they also consume a lot of time to recover a packet loss, leading to poor bandwidth utilization in the network. Unlike conventional TCP variants, this variant adjusts the rate of insertion of packets into network. It reduces the rate for conventional variants whereas increases the rate for high speed and long distance networks automatically [8].

2.2.9 TCP BIC: A congestion control algorithm, TCP BIC had been proposed by Lisong Xu et. al. It is a high speed congestion control protocol ensuring RTT fairness in large windows along with the maintenance of scalability as well as TCP friendliness. It combines the two ideas of additive increase and binary search increase. In case of large congestion window, additive increase is implemented with large increments. By this RTT fairness and scalability are maintained. In case of small congestion window, binary search increase is implemented and hence this maintains TCP friendliness [9].

2.2.10 TCP CUBIC: A congestion control algorithm TCP CUBIC had been proposed by Injong Rhee et. al. Its main feature is that its window growth function is defined on real time base system. Thus it makes the growth of window independent of RTT. Since window growth is not depending upon RTT, fairness of RTT is maintained as it does not hinder the flow of other RTT's. In this TCP, window growth function is a cubic function similar to TCP BIC [10].

2.2.11 Scalable TCP: This congestion control algorithm had been proposed by Tom Kelly. TCP congestion control algorithms works poorly in the high speed wide area networks as they have slow response time in such cases. Scalable TCP provides a robust mechanism which improves performance in high speed wide area network. It is designed to be incrementally deployable and performs similar to conventional TCP variants when the window size is small. Its main algorithm explains that when an ACK is received and

congestion is not detected, window size is increased. But if congestion is detected on ACK arrival, window size is decreased [12].

2.2.12 TCP Illinois: A congestion control algorithm, TCP Illinois had been proposed by Shaoliu. It utilizes the information regarding packet loss to pursue the action on the window size, whether the window size needs to be increased or decreased. It achieves high throughput, allocates network resources judicially and is compatible with standard TCPs. To achieve concave window curve, it keeps incremental factor large when there is no sign of congestion and it keeps incremental factor small if congestion control is likely to occur. Similarly it keeps decremental factor small when there is no sign of congestion and it keeps decremental factor large if congestion control is likely to occur [13].

2.2.13 Yeah TCP: This congestion control algorithm had been proposed by Andrea Baiocchi et.al. for high speed networks. It works on two modes: fast and slow. In fast mode it increments congestion window with respect to an aggressive rule. In slow mode, it works similar to TCP Reno [14].

2.3 Comparison of TCP Variants

Mazleena Salleh and Ahmad Zaki Abu Bakar (2005) compared four TCP variants- Tahoe, New-Reno, Vegas and SACK on the self-similar traffic. In this paper authors have concluded that even in small LAN such as FSKSM and UTM, the traffic flow demonstrates self-similar properties. Other conclusion was that TCP New-Reno performs best among the four variants in terms of efficiency and throughput [15].

A. R. Britto Pradeep et. al. (2011) performed comparisons of drop rates in various TCP variants such as Reno, New-Reno, STCP etc. on different routing protocols such as DSDV, DSR, AODV and TORA. The authors concluded in this paper that irrespective of the number of nodes and simulation time, TCP Tahoe had least packet drop [16].

Poonam Tomar and Prashant Panse (2011) had analyzed three TCP variants – Tahoe, Reno and Lite over the ad-hoc network on various parameters like packet loss, byte received, throughput and pause time. They concluded that no single TCP variant

performs well for all parameters. For few parameters, one variant had a better performance and for the other parameters, others performed well [17].

Ankur Lal and Sipy Dubey (2012) had analyzed the performance for TCP variants such as Reno, Vegas and NJplus with routing protocols such as AODV and DSDV over wireless networks. They concluded that no single TCP variant performs well for all parameters. For few parameters, one variant had a better performance and for the other parameters, others performed well. But TCP NJplus performance was little better than the rest compared variants [18].

Jawhar ben Abed et. al. (2012) compared eight low and high speed TCP variants with multiple flows. They concluded that some variants perform well for particular defined cases and perform poorly for other conditions. It was one of their observations that network architecture plays a significant role in the evaluation of performance of TCP variants [19].

Mohit P. Tahilianai et. al. (2013) analyzed high speed TCP variants for multi-hope non wired networks. They studied high speed variants by making changes in routing protocols such as DSDV- Destination Sequenced Distance Vector, AODV-Ad hoc On Demand Distance Vector and DSR- Dynamic Source Routing. They evaluated high speed variants on the basis of their throughputs for different topologies such as static and mobile topology. They made an observation that TCP's performance relies immensely on the type of routing protocol being used [20].

Madiha Kazmi et. al. (2014) has done the broad study of TCP variants in IP- Internet Protocol and MPLS- Multi-Protocol Label Switching network with CBR- Constant bit rate traffic. They analyzed different variants and came to the conclusion that TCP Vegas had the best performance and throughput among the variants as Reno, New- Reno, SACK, Tahoe and Vegas [21].

Kevin Fall and Sally Floyd had illustrates the benefits of adding selective acknowledgment to TCP. It deals with the comparison of the two versions of TCP Reno that is TCP New-Reno and TCP SACK. They performed various experiments and came to the conclusion that TCP SACK works better than TCP Reno and TCP New-Reno [22].

Table 2.1 shows comparison of various TCP Variants with respect to network type and key features that have been introduced by each variant.

Table 2.1: TCP Variant Comparison Table

SNO	TCP Variant	Network Type	Key Features
1	Tahoe	Wired	Slow Start, Congestion avoidance with AIMD, Coarse grain re-transmit timer.
2	Reno	Wired/Wireless	Fast retransmit, Reduce congestion window to half instead of 1
3	New-Reno	Wired/Wireless	Detect multiple packet loss
4	SACK	Wired/Wireless	Detect and acknowledge multiple packet loss.
5	Vegas	Wired/Wireless	Congestion avoidance mechanism.
6	Westwood	Wireless Networks	New faster recovery mechanism.
7	Veno	Wireless Networks	Detect type of loss congestion or random.
8	HTCP	High Speed and long distance Networks	Different incremental factor for different type of network.
9	BIC	Fast and Long distance Networks	BIC combine two schemes Additive increase and binary search increase.
10	Cubic	Fast and Long distance Networks	Enhances fairness property of BIC with cubic window growth function.
11	Hybla	Heterogeneous Networks	Solved RTT disparity problem, Implemented with sack option to recover multiple packet losses.
12	Scalable	High speed Wide area Network	Modified congestion window update algorithm which improves throughput in high speed and wide area networks.
13	Illinois	High Speed Network	Uses packet loss information to conclude whether window size to be increased or decreased and queuing delay to calculate the amount of increase or decrease.
14	Yeah	High Speed Network	Fast and slow as mode of working. In fast mode uses aggressive rule to increment congestion window and in slow mode it acts as TCP Reno.

Chapter 3

Problem Statement

Network has been experiencing fiery growth in the last few decades due to which traditional TCP is struggling to meet the demands of today's high bandwidth and low latency applications. With its focus on reliability and successful delivery at the expense of efficiency, TCP has become a limiting factor in the request to utilize and extract more value from the increased bandwidth and enhanced processing capabilities that define today's physical Internet.

To overcome drawbacks of traditional TCP, various TCP variants came where designed. Number of comparisons has been carried out between different TCP variants to get the best congestion control algorithm. Despite the huge variable usage of network, these comparisons did not take variable packet size into consideration. Today's network is not only used for sending text, rather images, high definition images, videos, high definition videos have become an important and major part of the network. The packet size varies for all such communication media. Even "Youtube" and "Whatapp" which are considered to be the most popular applications of these days are based upon TCP.

Therefore, a detailed analysis is required in order to gain an insight of these factors that determine the performance of the TCP variants. More specifically, it would be important to study individual factors affecting the performance of various TCP variants considering variable packet size as a key parameter.

The scope of this work is to analyze some existing TCP variants with respect to the comparison factors such as variable packet size, buffer size and bandwidth. The focus of this study is particularly around the performance metrics such as throughput, packet delay, jitter and packet drop rate. The major contribution of this research is to identify a TCP variant which yields good results for maximum number of performance metrics with respect to the various comparison factors.

Chapter 4

Experimentation

4.1 Topology Used

The topology on which we have performed various experiments is Dumbbell Topology consisting of 6 hosts and 2 routers shown in figure 4.1. The positive edge of this topology is that it is highly scalable. The traffic flow in this topology is analyzed by different TCP variants.

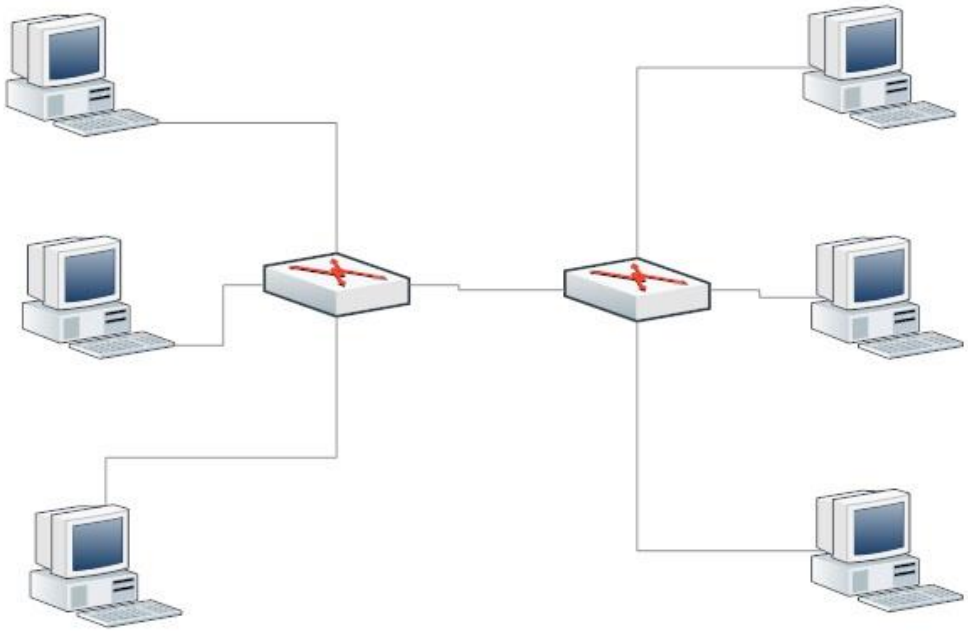


Figure 4.1: Dumb bell Topology

4.2 Simulation Environment

The results evaluated in this thesis are based upon the simulation which is done on NS2.35. It is a discrete event simulator. In NS2 mainly two languages are used C++ and object oriented tool command language (OTcl). In figure 4.2 **,** OTcl script file (.tcl) and traffic trace file (.bin) are taken as two inputs in NS2. Two output files are generated after the procedure in NS2 named as Network animator file (.nam) and ASCII trace file

(.tr). Nam file is used by Network animator whereas ASCII trace file is analysed with the help of different scripts such as python, pearl, awk etc to generate the final results[25,26,27,28].

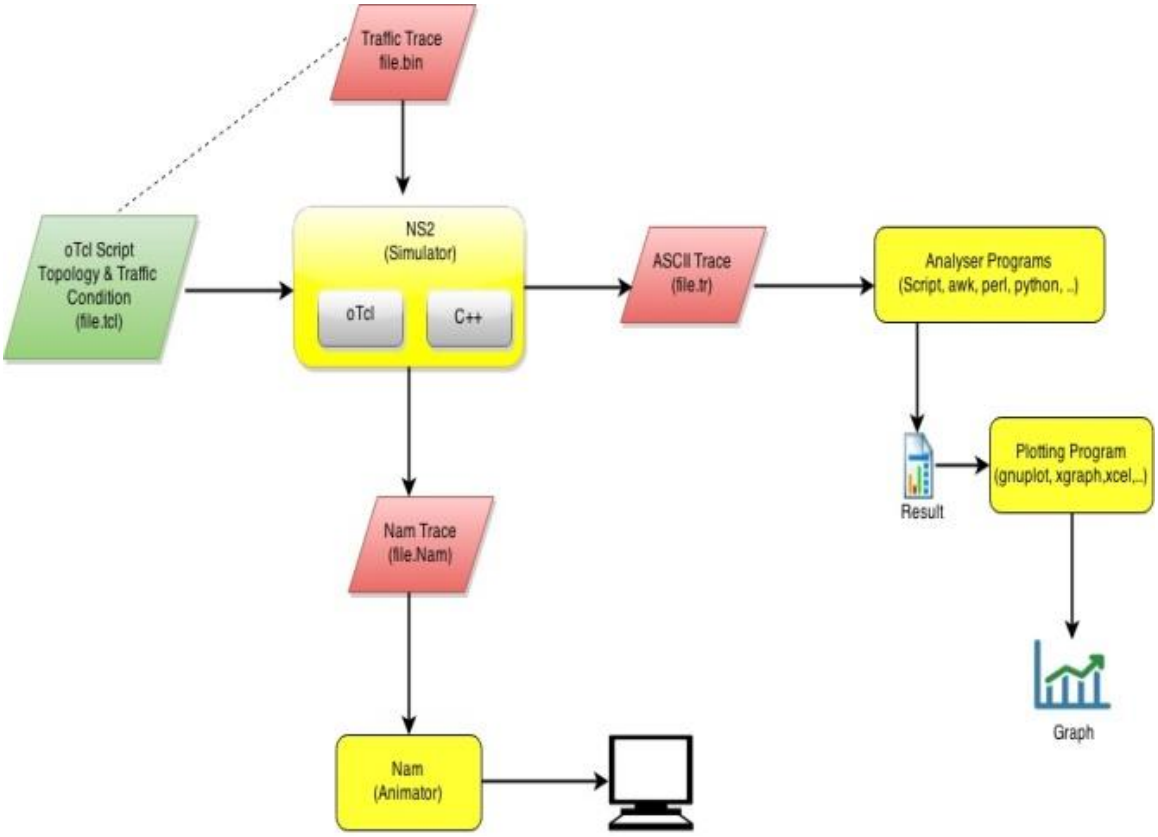


Figure 4.2: NS2 Architecture

4.3 Simulation Methodology

Performance metrics used in our experiments are:

4.3.1 Throughput: It is a measure to calculate the number of successfully transmitted packet from source to destination per unit time. It is compiled as follows.

$$\text{Throughput} = \frac{\text{Total number of packets sent}}{\text{Total transmission time}}$$

4.3.2 Packet loss: It is a measure for total number of packets that are discarded or lost while transmitting the data from source to its destination. It is calculated by

subtracting the total number of packets received at the receiver side from the total number of packets sent from sender side.

Packet Loss = Total number of packets received – Total number of packets sent

4.3.3 Average packet delay: It is the total end to end delay which is computed by addition of all delays that occur during network operation. Using the existing formula average delay would be given by the formula-

Average Packet Delay = processing delay + transmission delay + queueing delay + propagation delay

4.3.4 Jitter: It is the variation in latency between the arrival of packets which may occur due to the congestion in the network, time drift or any change in the route.

Table 4.1: Simulation Methodology

Attribute	Values	Type
Bandwidth	1 to 5 Mbps	Variable
Number of nodes	6	Fixed
Number of routers	2	Fixed
Buffer Size	10 to 50	Variable
Packet Size	2000	Variable
Traffic Type	FTP	Fixed
Link Type	Duplex	Fixed
Link Processing delay	1ms	Fixed
AQM	Drop Tail	Fixed

Chapter 5

Performance Evaluation

5.1 Variation in Packet Size

When packet size is varied in NS2, different behavior of the TCP variants was observed for various performance metrics and animation for the same was observed in Network Animator (Nam) which required .nam file as input generated from .tcl file as show in the figure 5.1.

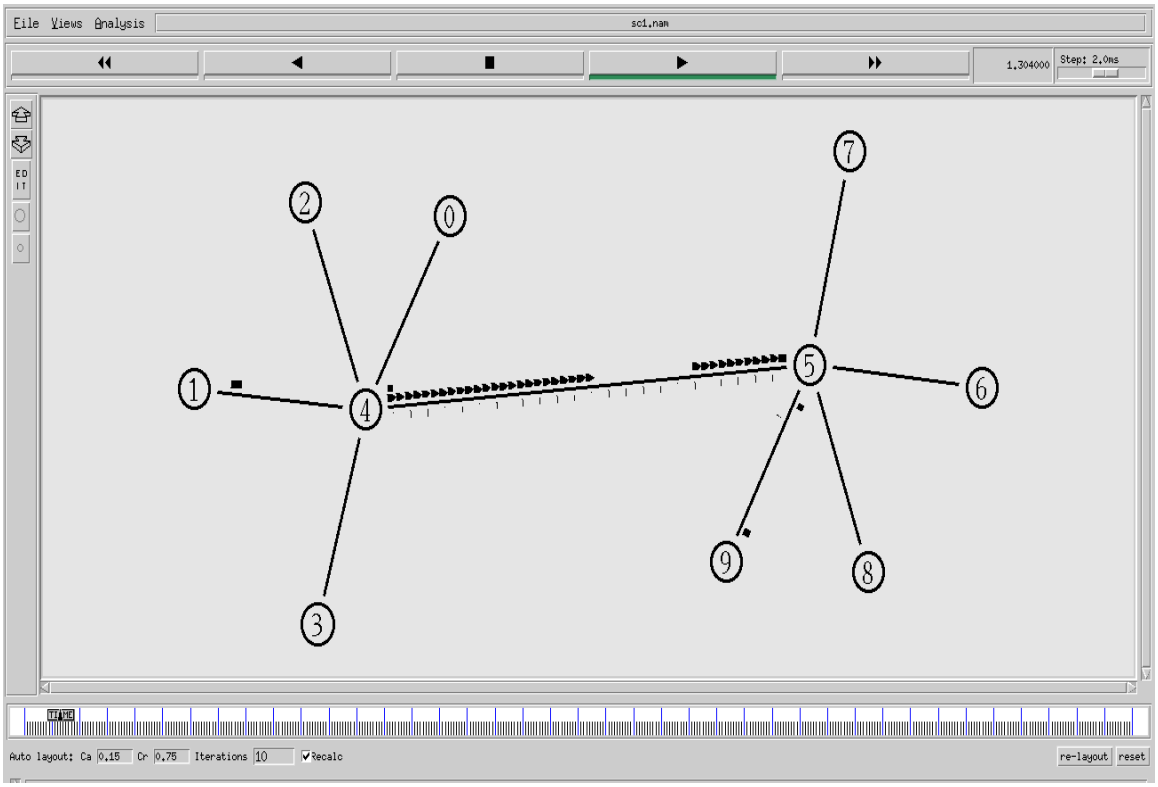
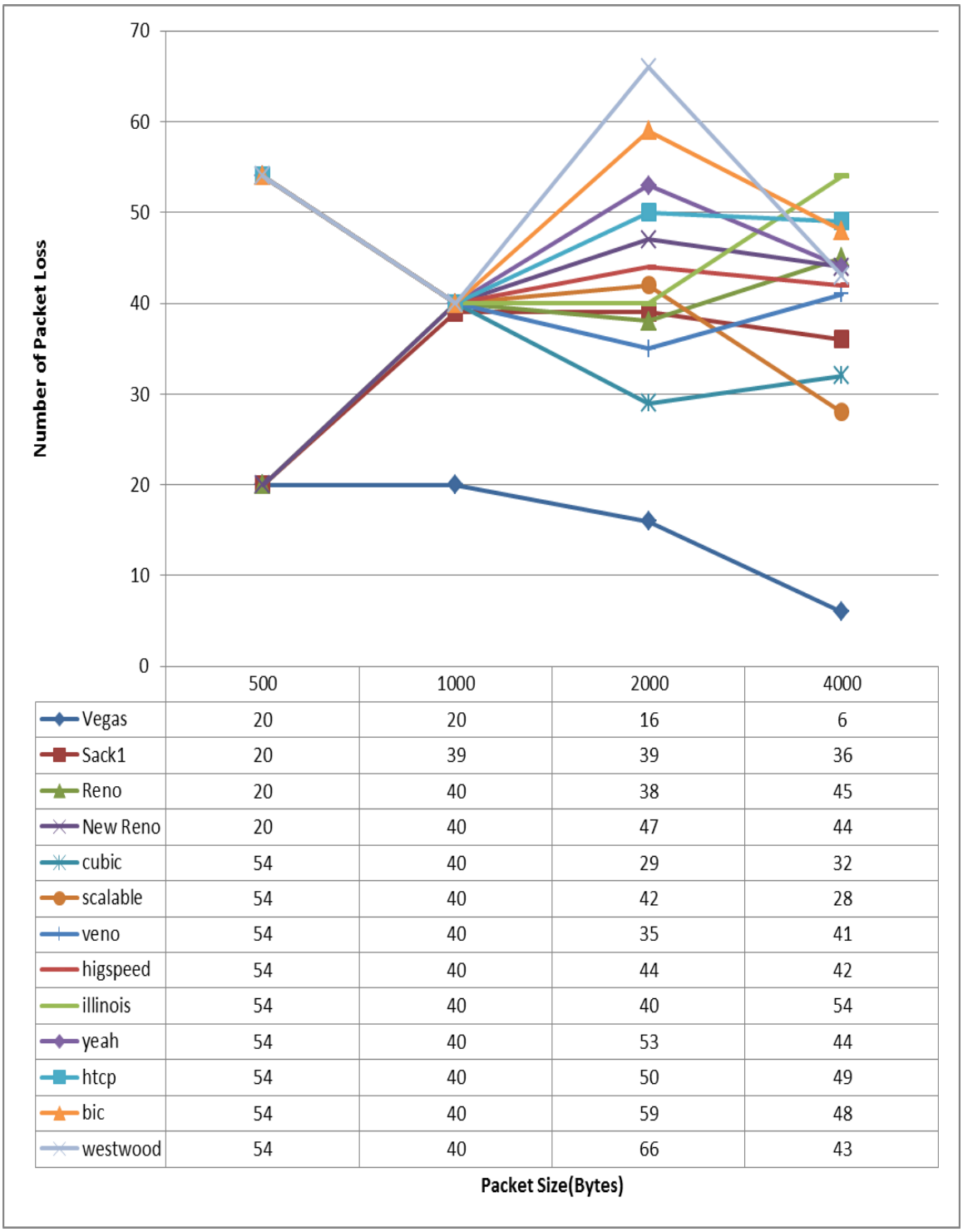


Figure 5.1: Dumbbell Topology in Nam

5.1.1 Packet loss: After performing the variation in packet size, it was observed that TCP Vegas had the best performance by achieving a minimum packet drop rate since it uses proactive measure to reduce congestion window before actually congestion occurs. Whereas TCP Westwood had the worst performance by having the maximum packet drop.

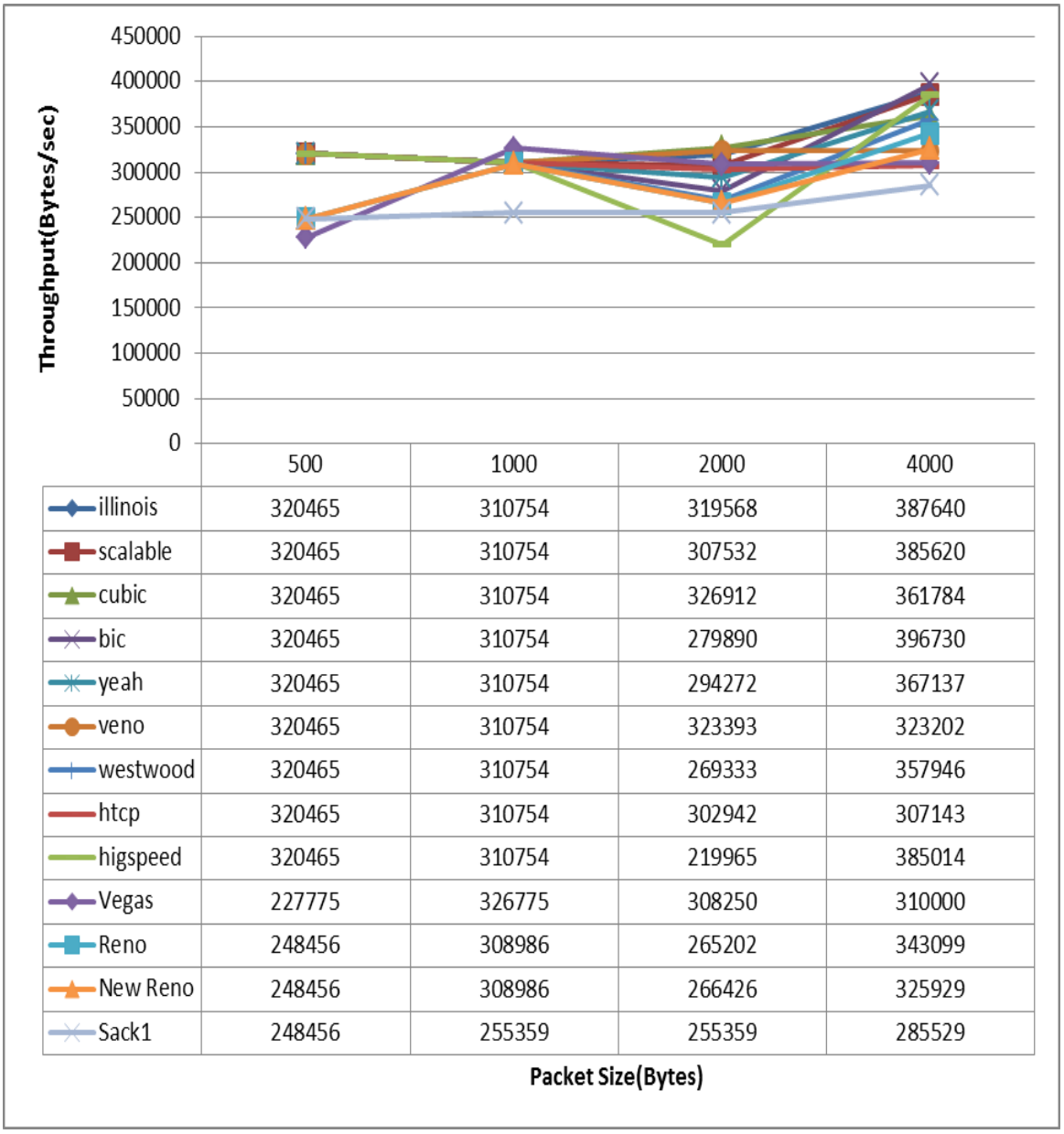


	500	1000	2000	4000
Vegas	20	20	16	6
Sack1	20	39	39	36
Reno	20	40	38	45
New Reno	20	40	47	44
cubic	54	40	29	32
scalable	54	40	42	28
veno	54	40	35	41
higspeed	54	40	44	42
illinois	54	40	40	54
yeah	54	40	53	44
htcp	54	40	50	49
bic	54	40	59	48
westwood	54	40	66	43

Figure 5.2: Packet Loss in variable packet size

5.1.2 Throughput: After performing the variation in packet size, it was found that TCP Illinois had the best performance by achieving a maximum throughput rate since it adjust

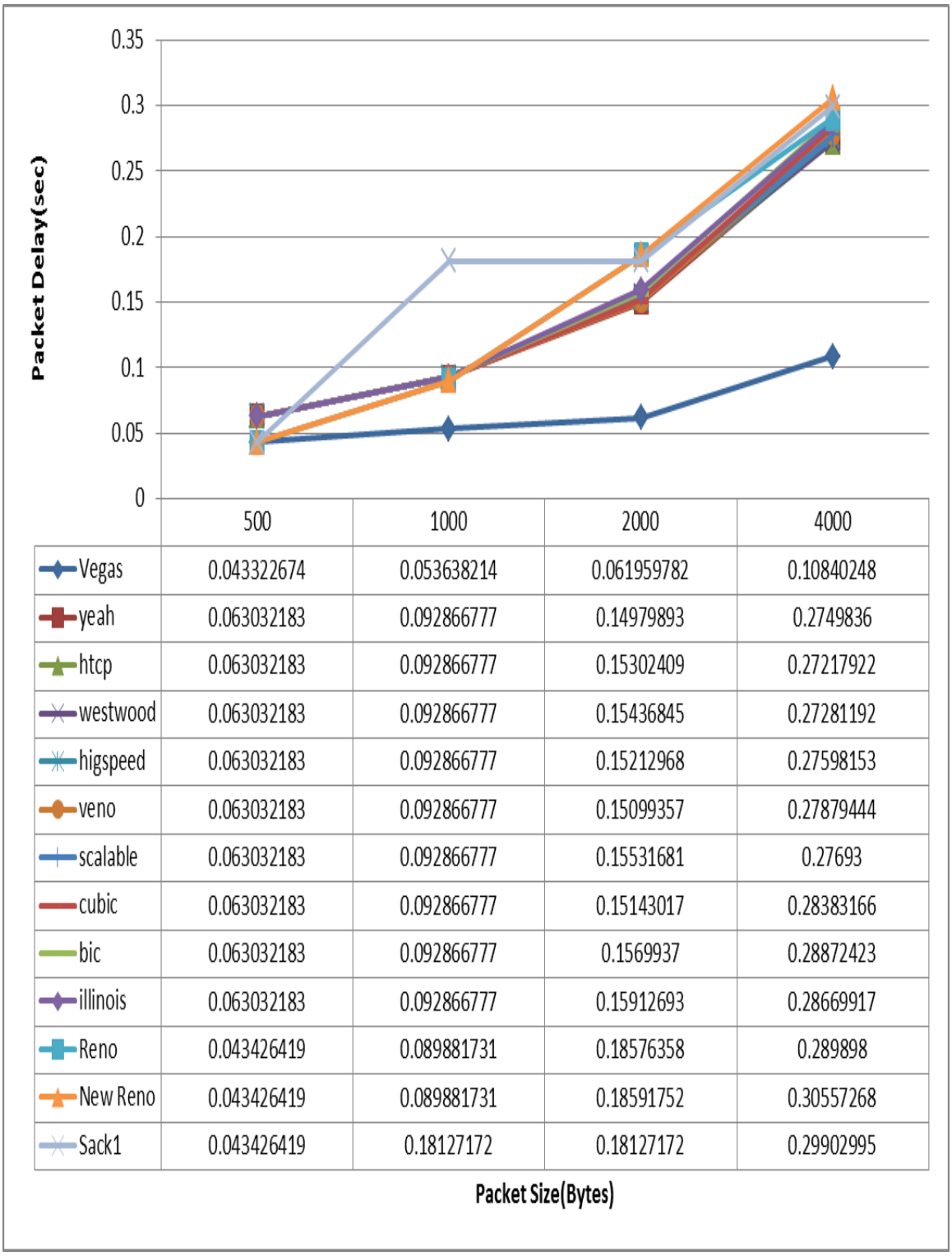
window size dynamically with respect to packet loss. Whereas TCP SACK had the worst performance by having a minimum throughput rate.



	500	1000	2000	4000
illinois	320465	310754	319568	387640
scalable	320465	310754	307532	385620
cubic	320465	310754	326912	361784
bic	320465	310754	279890	396730
yeah	320465	310754	294272	367137
veno	320465	310754	323393	323202
westwood	320465	310754	269333	357946
htcp	320465	310754	302942	307143
higspeed	320465	310754	219965	385014
Vegas	227775	326775	308250	310000
Reno	248456	308986	265202	343099
New Reno	248456	308986	266426	325929
Sack1	248456	255359	255359	285529

Figure 5.3: Throughput in variable packet size

5.1.3 Packet Delay: After performing the variation in packet size, it was found that TCP Vegas had the best performance by achieving a minimum packet delay since it uses proactive measure to reduce congestion window before actually congestion occurs. Whereas TCP SACK had the worst performance by having the maximum packet delay.

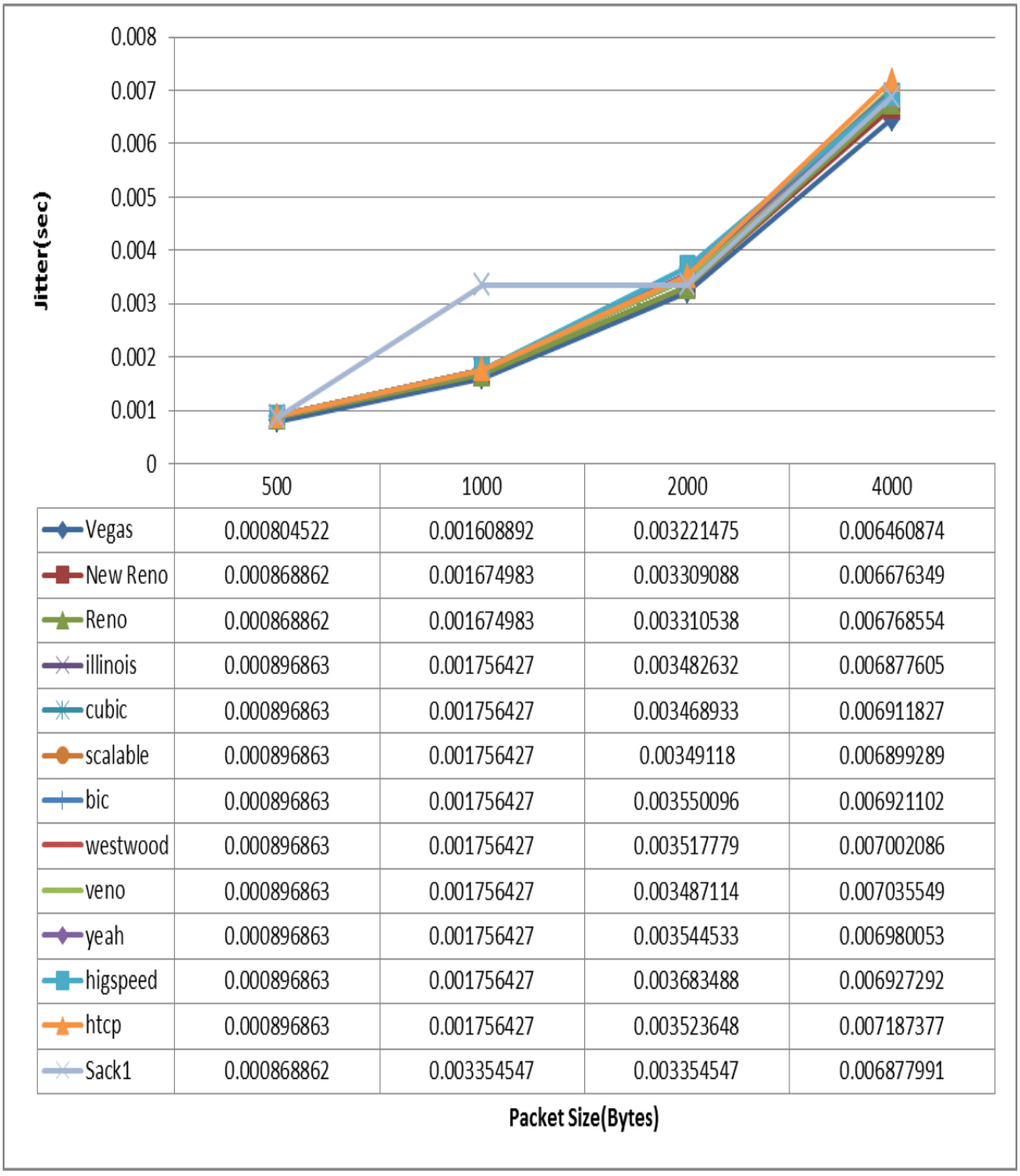


	500	1000	2000	4000
Vegas	0.043322674	0.053638214	0.061959782	0.10840248
yeah	0.063032183	0.092866777	0.14979893	0.2749836
htcp	0.063032183	0.092866777	0.15302409	0.27217922
westwood	0.063032183	0.092866777	0.15436845	0.27281192
higspeed	0.063032183	0.092866777	0.15212968	0.27598153
veno	0.063032183	0.092866777	0.15099357	0.27879444
scalable	0.063032183	0.092866777	0.15531681	0.27693
cubic	0.063032183	0.092866777	0.15143017	0.28383166
bic	0.063032183	0.092866777	0.1569937	0.28872423
illinois	0.063032183	0.092866777	0.15912693	0.28669917
Reno	0.043426419	0.089881731	0.18576358	0.289898
New Reno	0.043426419	0.089881731	0.18591752	0.30557268
Sack1	0.043426419	0.18127172	0.18127172	0.29902995

Figure 5.4: Packet Delay in variable packet size

5.1.4 Jitter: After performing the variation in packet size, it was observed that TCP Vegas had the best performance by achieving a minimum jitter since it uses proactive

measure to reduce congestion window before actually congestion occurs.. Whereas TCP SACK had the worst performance by having the maximum jitter.



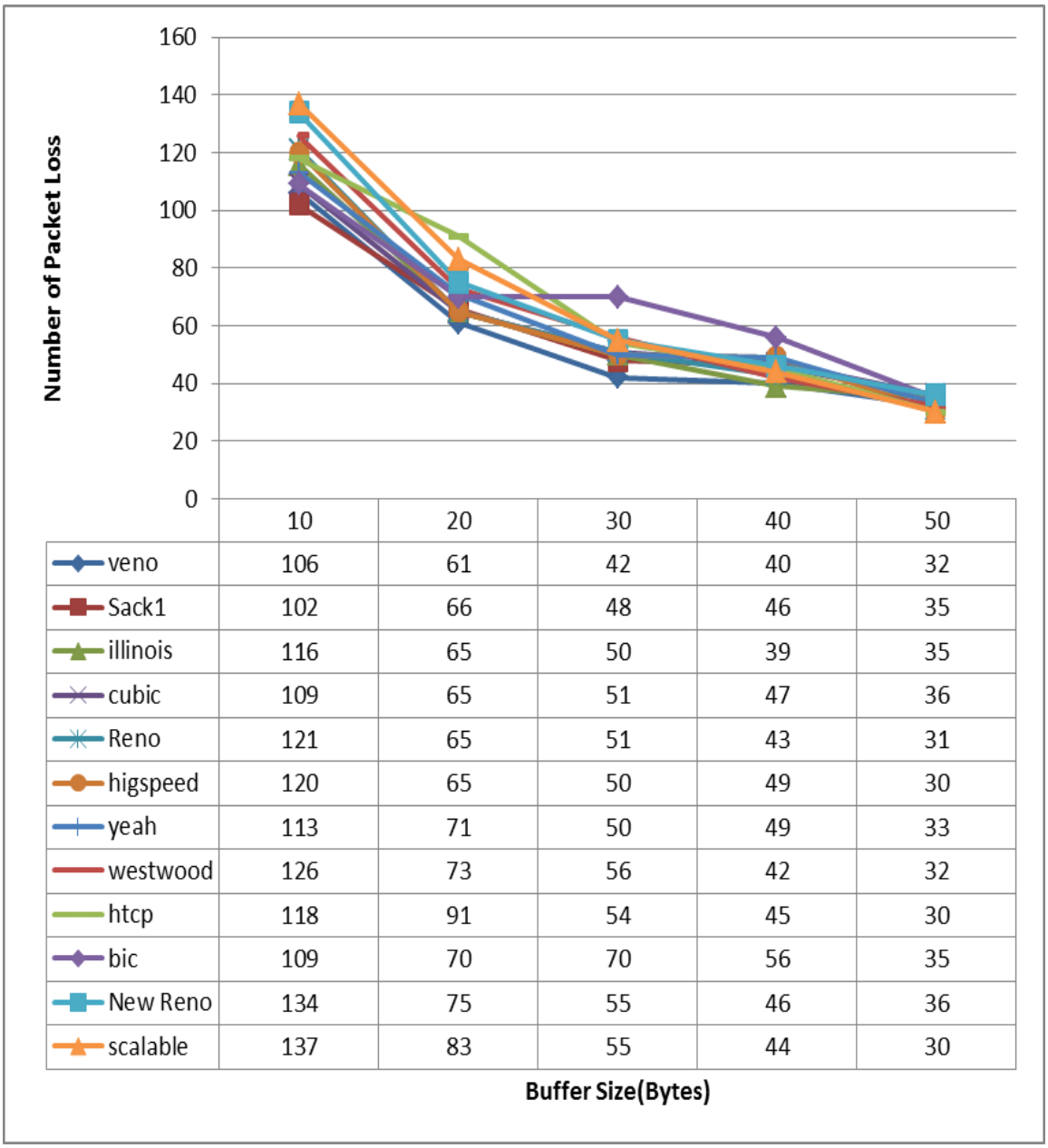
	500	1000	2000	4000
Vegas	0.000804522	0.001608892	0.003221475	0.006460874
New Reno	0.000868862	0.001674983	0.003309088	0.006676349
Reno	0.000868862	0.001674983	0.003310538	0.006768554
illinois	0.000896863	0.001756427	0.003482632	0.006877605
cubic	0.000896863	0.001756427	0.003468933	0.006911827
scalable	0.000896863	0.001756427	0.00349118	0.006899289
bic	0.000896863	0.001756427	0.003550096	0.006921102
westwood	0.000896863	0.001756427	0.003517779	0.007002086
veno	0.000896863	0.001756427	0.003487114	0.007035549
yeah	0.000896863	0.001756427	0.003544533	0.006980053
higspeed	0.000896863	0.001756427	0.003683488	0.006927292
htcp	0.000896863	0.001756427	0.003523648	0.007187377
Sack1	0.000868862	0.003354547	0.003354547	0.006877991

Figure 5.5: Jitter in variable packet size

5.2 Variation in Buffer Size

When packet size is varied in NS2, different behavior of the TCP variants was observed for various performance metrics.

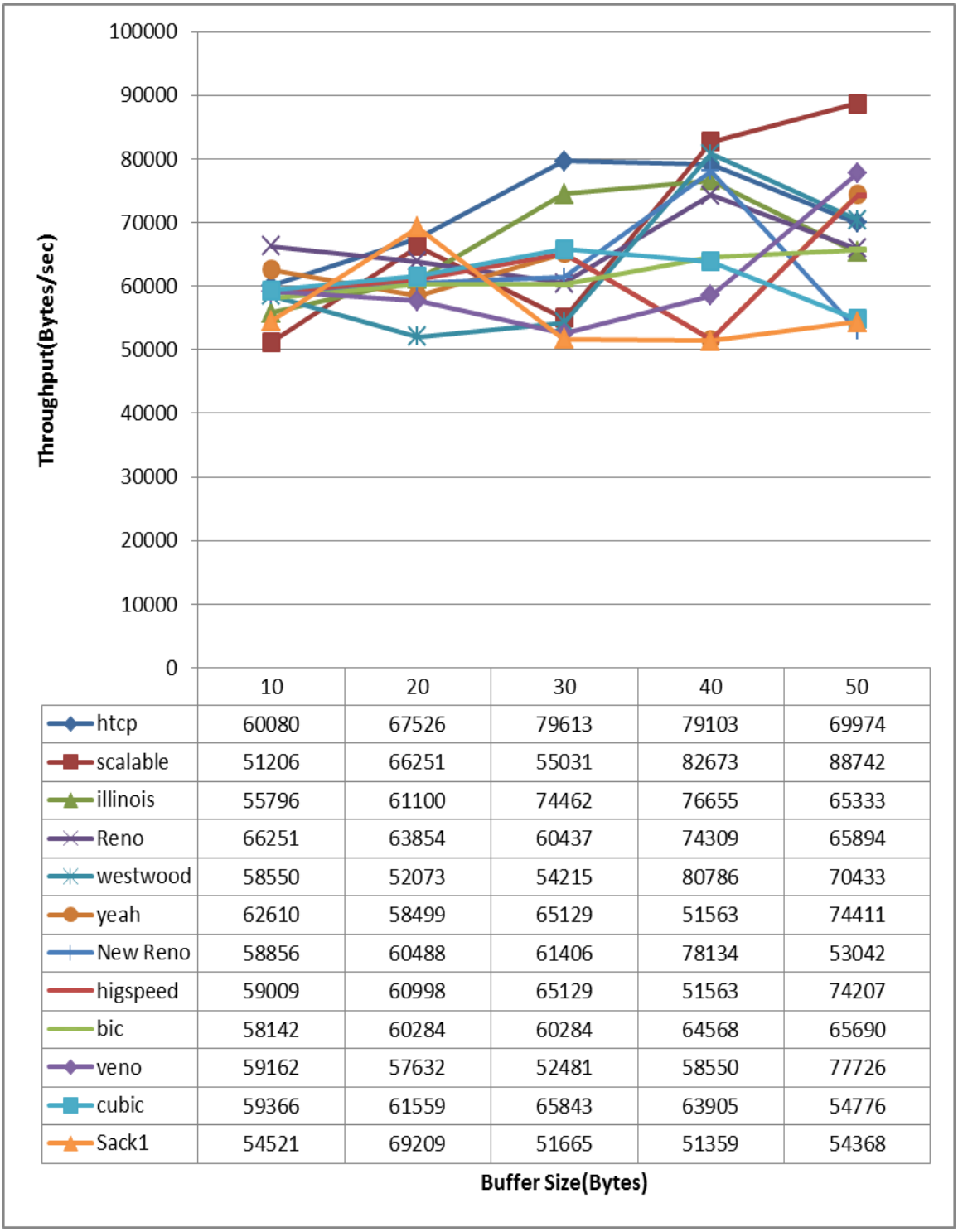
5.2.1 Packet Loss: After performing the variation in packet size, it was concluded that TCP Veno had the best performance by achieving a minimum packet drop rate. Whereas TCP Scalable had the worst performance by having the maximum packet drop.



	10	20	30	40	50
veno	106	61	42	40	32
Sack1	102	66	48	46	35
illinois	116	65	50	39	35
cubic	109	65	51	47	36
Reno	121	65	51	43	31
higspeed	120	65	50	49	30
yeah	113	71	50	49	33
westwood	126	73	56	42	32
htcp	118	91	54	45	30
bic	109	70	70	56	35
New Reno	134	75	55	46	36
scalable	137	83	55	44	30

Figure 5.6: Packet loss in variable buffer size

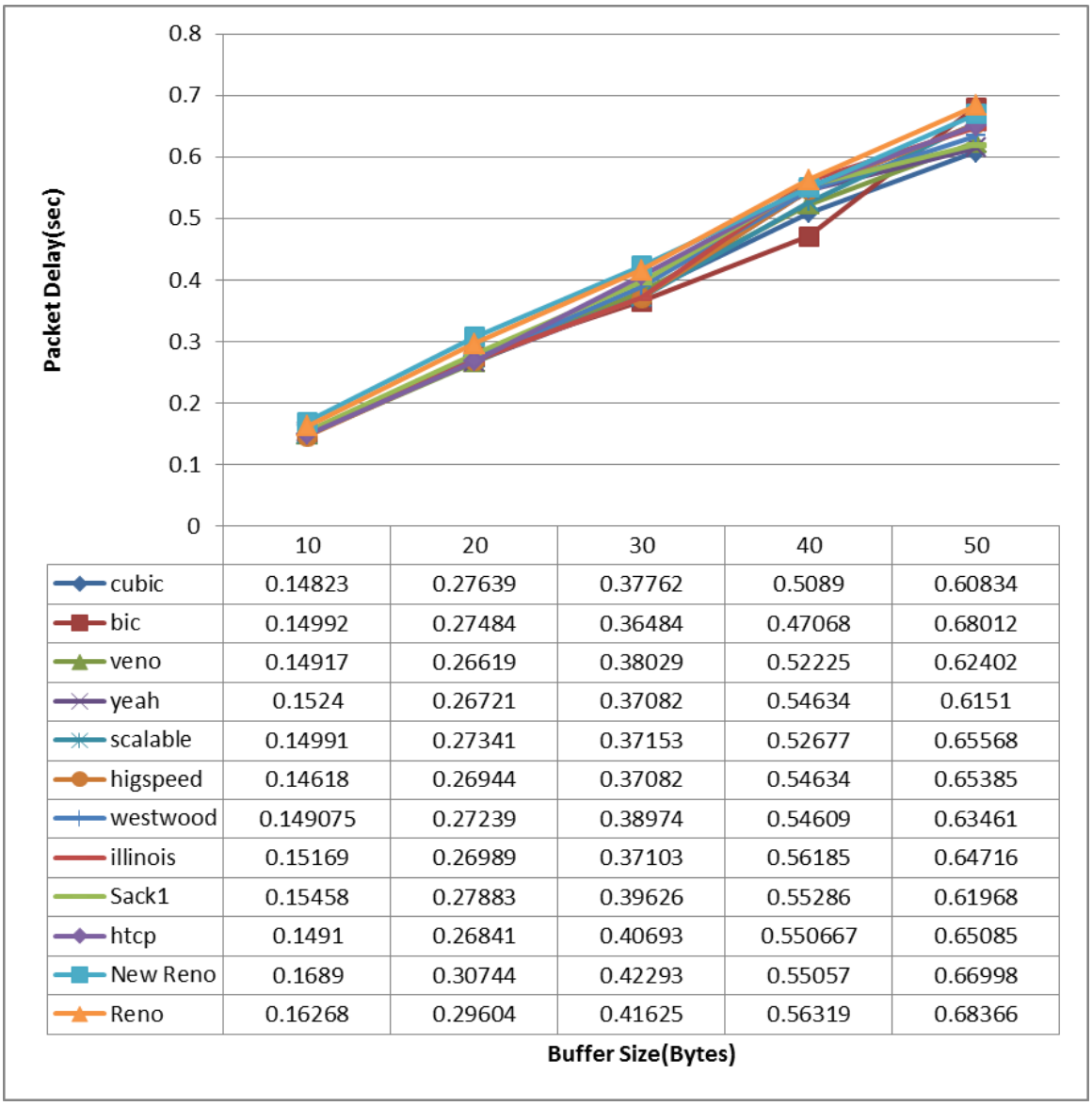
5.2.2 Throughput: After performing the variation in packet size, it was observed that TCP htcp had the best performance by achieving a maximum throughput rate. Whereas TCP SACK had the worst performance by having a minimum throughput rate.



	10	20	30	40	50
htcp	60080	67526	79613	79103	69974
scalable	51206	66251	55031	82673	88742
illinois	55796	61100	74462	76655	65333
Reno	66251	63854	60437	74309	65894
westwood	58550	52073	54215	80786	70433
yeah	62610	58499	65129	51563	74411
New Reno	58856	60488	61406	78134	53042
higspeed	59009	60998	65129	51563	74207
bic	58142	60284	60284	64568	65690
veno	59162	57632	52481	58550	77726
cubic	59366	61559	65843	63905	54776
Sack1	54521	69209	51665	51359	54368

Figure 5.7: Throughput in variable buffer size

5.2.3 Packet Delay: After performing the variation in packet size, it was found that TCP Cubic had the best performance by achieving a minimum packet delay. Whereas TCP Reno had the worst performance by having the maximum packet delay.



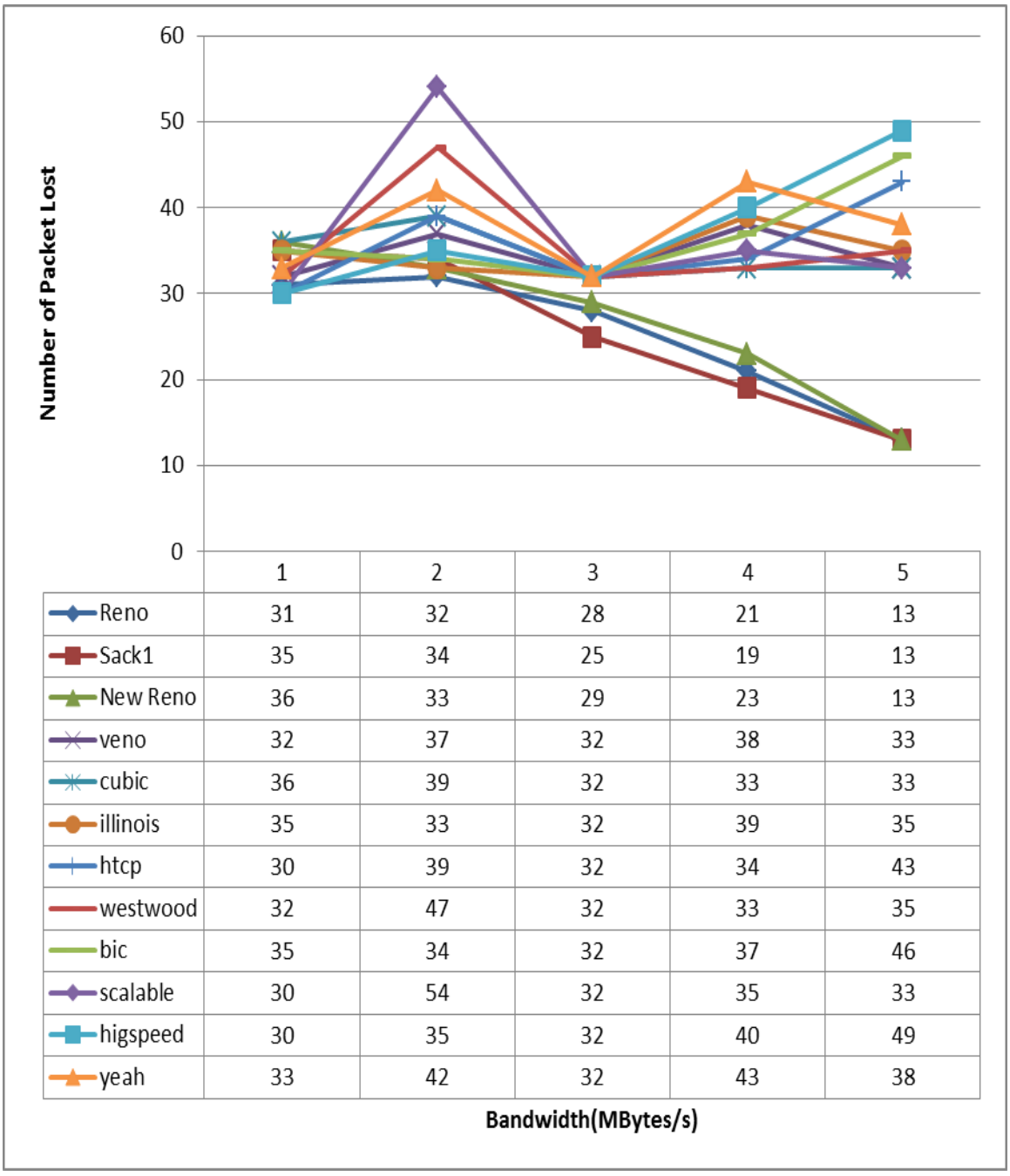
	10	20	30	40	50
cubic	0.14823	0.27639	0.37762	0.5089	0.60834
bic	0.14992	0.27484	0.36484	0.47068	0.68012
veno	0.14917	0.26619	0.38029	0.52225	0.62402
yeah	0.1524	0.26721	0.37082	0.54634	0.6151
scalable	0.14991	0.27341	0.37153	0.52677	0.65568
higspeed	0.14618	0.26944	0.37082	0.54634	0.65385
westwood	0.149075	0.27239	0.38974	0.54609	0.63461
illinois	0.15169	0.26989	0.37103	0.56185	0.64716
Sack1	0.15458	0.27883	0.39626	0.55286	0.61968
htcp	0.1491	0.26841	0.40693	0.550667	0.65085
New Reno	0.1689	0.30744	0.42293	0.55057	0.66998
Reno	0.16268	0.29604	0.41625	0.56319	0.68366

Figure 5.8: Packet loss in variable buffer size

5.3 Variation in Bandwidth

When packet size is varied in NS2, different behavior of the TCP variants was observed for various performance metrics.

5.3.1 Packet Loss: After performing the variation in packet size, it was found that TCP Reno had the best performance by achieving a minimum packet drop rate. Whereas TCP Yeah had the worst performance by having the maximum packet drop.

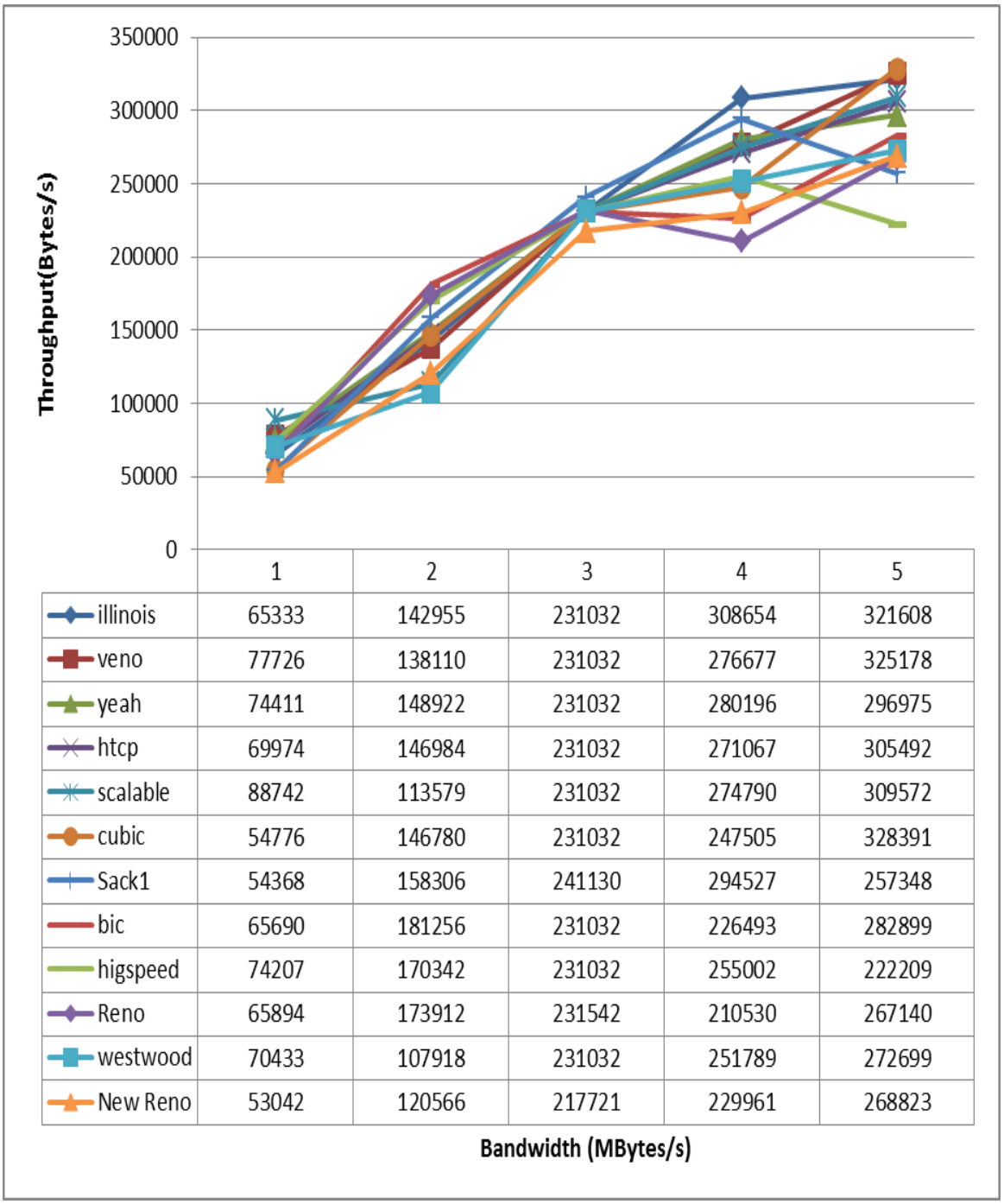


	1	2	3	4	5
Reno	31	32	28	21	13
Sack1	35	34	25	19	13
New Reno	36	33	29	23	13
veno	32	37	32	38	33
cubic	36	39	32	33	33
illinois	35	33	32	39	35
htcp	30	39	32	34	43
westwood	32	47	32	33	35
bic	35	34	32	37	46
scalable	30	54	32	35	33
higspeed	30	35	32	40	49
yeah	33	42	32	43	38

Figure 5.9: Packet loss in variable bandwidth

5.3.2 Throughput: After performing the variation in packet size, it was observed that TCP Illinois had the best performance by achieving a maximum throughput rate.

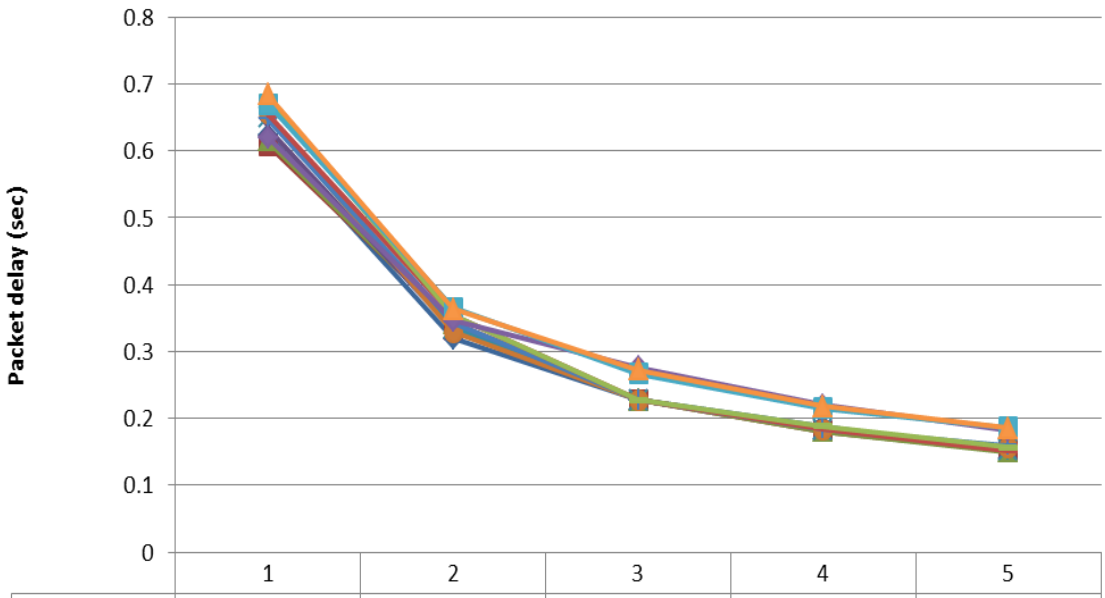
Whereas TCP New-Reno had the worst performance by having a minimum throughput rate.



	1	2	3	4	5
illinois	65333	142955	231032	308654	321608
veno	77726	138110	231032	276677	325178
yeah	74411	148922	231032	280196	296975
htcp	69974	146984	231032	271067	305492
scalable	88742	113579	231032	274790	309572
cubic	54776	146780	231032	247505	328391
Sack1	54368	158306	241130	294527	257348
bic	65690	181256	231032	226493	282899
higspeed	74207	170342	231032	255002	222209
Reno	65894	173912	231542	210530	267140
westwood	70433	107918	231032	251789	272699
New Reno	53042	120566	217721	229961	268823

Figure 5.10: Throughput in variable bandwidth

5.3.3 Packet Loss: After performing the variation in packet size, it was observed that TCP Veno had the best performance by achieving a minimum packet delay. Whereas TCP Reno had the worst performance by having the maximum packet delay.



	1	2	3	4	5
veno	0.62402	0.31902	0.22769	0.18815	0.15099
cubic	0.60834	0.34361	0.22769	0.1806	0.15143
yeah	0.6151	0.3414	0.22769	0.18086	0.14979
westwood	0.63461	0.3323	0.22769	0.18297	0.15436
htcp	0.65085	0.33477	0.22769	0.18308	0.15302
scalable	0.65568	0.32878	0.22769	0.18347	0.15531
illinois	0.64716	0.34351	0.22769	0.1845	0.15912
higspeed	0.65385	0.35417	0.22769	0.18391	0.15212
bic	0.68012	0.35447	0.22769	0.18781	0.15699
Sack1	0.61968	0.34651	0.2764	0.22011	0.18127
New Reno	0.66998	0.36527	0.26663	0.2148	0.18591
Reno	0.68366	0.36358	0.27215	0.21769	0.18576

Bandwidth(MByte/s)

Figure 5.11: Packet Delay in variable bandwidth

Chapter 6

Conclusion and Future Work

Congestion Control is a significant issue in fixed and mobile networks. Different TCP variants such as Tahoe, Reno, New-Reno, Vegas, SACK etc. were compared for different metrics such as variable packet size, buffer size and bandwidth. We have studied the behavior and performance of TCP variants over various performance metrics such as throughput, jitter, packet loss and packet delay over the dumbbell topology. While analyzing the variable packet size, it was found that TCP Vegas has the best performance over packet loss, packet delay and jitter. When variable packet size was analyzed over throughput, it was found that TCP SACK gave the best performance. While variable buffer size was analyzed over packet loss, it was found that TCP VENO has the best performance whereas over packet delay, TCP Cubic observed to give the best performance. When variable buffer size was analyzed over throughput, it was found that TCP HTP gave the best performance. While variable bandwidth was analyzed over packet loss, it was found that TCP Reno has the best performance whereas over packet delay, TCP Veno observed to give the best performance. When variable bandwidth was analyzed over throughput, it was found that TCP New-Reno gave the best performance. Thus we can conclude no single algorithm could overcome congestion and unpredictable behavior of network. Deciding factors for any TCP variant to be effective or non-effective are the parameters on which simulation is carried out.

In future this area can be explored to a higher level for the development of a new algorithm for congestion control and avoidance which can handle unpredictable behavior of the network to improve TCP performance.

References

[1] V. Jacobson. "Congestion Avoidance and Control", *ACM SIGCOMM computer communication review*, vol. 18, no. 4, pp. 314-329, 1988.

[2] V. Jacobson. "Modified TCP Congestion Control and Avoidance Algorithms", [Online]. Available: ftp://ftp.isi.edu/end2end/end2end-interest-1990.mail. [Accessed: Mar. 28, 2015].

[3] S. Floyd and T. Henderson. "The New Reno modification to TCP's Fast recovery algorithm", *Request for Comment 2582-IETF*, April 1999.

[4] M. Mathis and J. Mahdavi. "TCP selective acknowledgment options", *Request for Comment 2018-IETF*, October 1996.

[5] L. S. Brakmo, L. L. Peterson. "TCP Vegas-End to end congestion avoidance on a global internet", *IEEE Journal on Selected Areas in Communications*, vol. 13, no.8, pp. 1465–1480, 1995.

[6] C. C. Westwood, M. Gerla, S. Mascolo, M. Y. Sanadidi and R. Wang. "TCP westwood: Bandwidth Estimation for enhanced Transport Over Wireless Links", *ACM Mobicom,* vol. 8, no. 5, pp. 287–297, July 2001.

[7] C. P. Fu. "TCP Veno: End-to-End Congestion Control Over Heterogeneous Networks", Ph.D. dissertation, The Chinese University, Hong Kong, 2001.

[8] R. Shorten and D. Leith. "H-TCP: TCP for High-Speed and Long-Distance Networks", *Second International Workshop on Protocols for Fast Long-Distance Networks*, Argonne, Illinois USA, February 16-17, 2004.

[9] L. Xu, K. Harfoush and I. Rhee. "Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks", *Proceeding of IEEE INFOCOM 2004*, March 2004.

[10] I. Rhee and L. Xu. "CUBIC: A New TCP-Friendly High-Speed TCP Variant", *Proceeding of PFLDnet*, France, February 2005.

[11] C. Caini and R. Firrincieli. “TCP Hybla: A TCP enhancement for heterogeneous networks”, *International journal of satellite communications and networking*, vol. 22, no. 5, pp. 547–566, 2004.

[12] T. Kelly. "Scalable TCP: Improving performance in highspeed wide area networks", *ACM SIGCOMM* Computer Communication Review, vol. 33, no. 2, pp. 83-91, April 2003.

[13] S. Liu, T. Basar and R. Srikant. “TCP-illinois: A delay and loss-based congestion control algorithm for high-speed networks”, *Performance Evaluation*, vol. 65, no. 6, pp. 417-440, 2008.

[14] A. Baiocchi, A. Castellani and F. Vacirca. "YeAH-TCP: Yet Another Highspeed TCP", *Proceedings of The fifth PFLDNET*, Los Angeles, CA, February 2007.

[15] M. Salleh and A. Z. Abu Bakar. "Comparative performance of TCP variants on self-similar traffic”, *Proceedings of the Postgraduate Annual Research Seminar*, 2005.

[16] A. R. Britto Pradeep, N. Dhinakaran and P. Angelin. "Comparison of Drop Rates in Different TCP Variants against Various Routing Protocols", *International Journal of Computer Applications*, vol. 20, no. 6, April 2011.

[17] P. Tomar and P. Panse. "A Comprehensive Analysis and Comparison of TCP Tahoe, TCP Reno and TCP Lite", *International Journal of Computer Science and Information Technologies*, vol. 2, no. 5, 2011.

[18] J. Ben, L. Sinda, M. Ali Mani and R. Mbarek. "Comparison of high speed congestion control protocols”, *International Journal of Network Security & Its Applications*, vol. 4, no. 5, pp. 15-24, September 2012.

[19] A. Lal and S. Dubey. "AODV, DSDV Performance Analysis with TCP Reno, TCP Vegas and TCP-NJplus Agents of Wireless Networks on Ns2”, *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 2, no. 7, July 2012.

[20] M. P. Tahiliani, K. C. Shet and T. G. Basavaraju. "Comparative Study of High-Speed TCP Variants in Multi-Hop Wireless Networks", *International Journal of Computer Theory and Engineering*, vol. 5, no. 5, pp. 802-806, October 2013.

[21] M. Kazmi, A. Shamim, N. Wahab and F. Anwar. "Comparison of TCP Tahoe, Reno, New Reno, Sack and Vegas in IP and MPLS Networks under Constant Bit Rate Traffic", *International Conference on Advanced Computational Technologies & Creative Media*, Pattaya, 2014, pp. 14-15.

[22] K. Fall and S. Floyd. "Simulation-based comparisons of Tahoe, Reno and SACK TCP", *ACM SIGCOMM Computer Communication Review*, vol. 26, no. 3, pp. 5-21, 1996.

[23] W. Stevens. "TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms", *Request for Comment* 2001-IETF, January 1997.

[24] K. Fall and S. Floyd. "Simulation-based comparison of Tahoe, Reno, and sack TCP", *Computer Communication Review*, vol. 26, no. 3, pp. 5–21, 1996.

[25] "NS-2, Network Simulator". [Online]. Available: www.isi.edu/nsnam/ns. [Accessed: Apr. 3, 2015].

[26] D. X. Wei and P. Cao. "NS-2 TCP-Linux: An NS-2 TCP implementation with congestion control algorithms from Linux", *Proceedings from the 2006 workshop on ns-2: the IP network simulator*, 2006.

[27] D. X. Wei and P. Cao. "A Linux TCP implementation for NS2". [Online]. Available: http://netlab.caltech.edu/projects/ns2tcplinux/ns2linux. [Accessed: Mar. 12, 2015].