

Enhancement of TCP Congestion Control Protocol

Submitted by:

Arina Islam Ahona

ID:2018-1-60-179

Umma Sahorina Sultana

ID:2018-1-60-145

Rehana Akter

ID:2018-1-60-202

Ehsanul Haque

ID:2018-1-68-079

Supervised by:

Dr. Anisur Rahman

Associate Professor

Department of Computer Science Engineering

East West University

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree

of Bachelor of Science in Computer Science and Engineering to the

Department of Computer Science and Engineering

East West University

Dhaka-1212, Bangladesh



Abstract

In Network congestion control is given importance so that we can get error free and high data rates. Transmission Control Protocol (TCP) is a protocol which was designed for the reliable delivery of the packets. We have used an algorithm called slow and start algorithm[1]. In this algorithm congestion window size plays a vital role to analyze the performance of the network. If the data is sent successfully and the acknowledgement is received successfully then the congestion window size is increased. So, the congestion window is increased depending on the condition of the network. When the congestion occurs, the congestion window is reduced[2]. In this paper we have tried to improve the slow start phase of the algorithm because in slow start phase the congestion window size increases exponentially but when the sender has to send a large segment of data immediately and the condition of the network is also good as we are using slow and start algorithm, we have to start with small segment which is a waste of time and also waste of bandwidth. Therefore, we have inserted a timer of 26ms depending upon which we will decide how many segments we can send. We have also tried to show the delivery of the packets using Network simulator (NS2).

This paper is a theoretical study of the mechanisms to control the congestion of the network to save the packets from being dropped. Our focus is to use the slow and start algorithm to control the congestion.

Declaration

We hereby declare that this project is completed under CSE497 and has not been submitted elsewhere for requirement of any degree or diploma or for any purpose except for publication.

Arina Islam Ahona

ID: 2018-1-60-179

Department of Computer Science and Engineering

East West University

Umma Sahorina Sultana

ID: 2018-1-60-145

Department of Computer Science and Engineering

East West University

Rehana Akter

ID: 2016-1-60-202

Department of Computer Science and Engineering

East West University

Ehsanul Haque

ID: 2018-1-68-079

Department of Computer Science and Engineering

East West University

Letter of Acceptance

We, hereby declare that thesis is from the student's own work and best effort of us, and all other sources of information used are acknowledged. This thesis is submitted with our approval.

Supervisor

Dr. Anisur Rahman

Associate Professor

Department of Computer Science and Engineering

East West University

Chairperson

Dr. Taskeed Jabid

Associate Professor and Chairperson

Department of Computer Science and Engineering

East West University

Acknowledgement

We begin by praising Almighty Allah since all the praises due to him. It is His mercy that made our work and life easier.

Our sincere gratitude goes to our parents and family members, their unbreakable trust in us and continuous support made our life beautiful.

We express special thanks to our friends and seniors whose suggestions and helpful attitude made our thesis more efficient and dependable.

Dr. Anisur Rahman, our supervisor, his considerate and friendly attitude, apt advice, unique direction, and inspiration through the time of our research, helped us a lot.

We thank our teachers of CSE department in East West University because of their helpful attitude.

At last, a grateful thanks for the researchers in the field of Wireless Network, Congestion Control Congestion Control Routing System (CCRS).

Abbreviation and Acronyms

TCP	Transmission Control Protocol
QoS	Quality of Service
CCRS	Congestion Control Routing System
GAIMD	General Additive Increase and Multiplicative Decrease
RTOA	Roundtrip Time of Arrival
TL	Transmission Loss
RSS	Received Signal Strength
RRT	Round Routing Time
NAM	Network Animator
CWND	Congestion Window

Table of Contents

Declaration of Authorship	01
Abstract	02
Declaration	03
Letter of Acceptance	05
Acknowledgement	06
Abbreviation and Acronyms	07
Table of contents	08
List of Figures	10
Chapter 1 Introduction	11
Chapter 2 Literature Review	13
Chapter 3 Methodology	
3.1 TCP Tahoe	20
3.2 TCP Reno	21
3.3 TCP New Reno	21
3.4 TCP Vegas	22
3.5 Delay Based Congestion Control Algorithms	22
3.6 Rate-Based Congestion Control Scheme	23
3.7 The TCP Slow-Strat algorithm	25
3.8 Congestion Detection Phase	28
3.9 Pseudocode	30

3.10 The Imbalance Scenery of TCP	32
3.11 The Problem of Slow-Start Algorithm	34
3.12 Our Proposed Method	34
Chapter 4 Simulation & Results	
4.1 Network Simulator	37
4.2 Simulation	43
4.3 Code of the Proposed Method	44
4.4 AWK file of total packet Calculation	48
4.5 AWK file of Throughput and Latency Calculation	49
4.6 Simulated Result	50
4.7 Trace file	51
4.8 Graph	52
Chapter 5 Conclusion and Future Work	53
Chapter 6 References	54

List of Figures

Figure 1: Different packet tracks coming to main router	11
Figure 2: Architecture of the Existing System (Sharadehandra et al.2007)	13
Figure 3: Flow Chart of Slow Start Algorithm	25
Figure 4: Slow Start Process	26
Figure 5: Congestion Avoidance Chart	27
Figure 6: TCP Congestion Policy Chart	29
Figure 7: Graph of Transmission x Congestion Window	29
Figure 8: Initial Flow Chart of CCRS	31
Figure 9: Describing capacity of network when high	32
Figure 10: Graph of Packet sent x Packet delivered with respect to maximum capacity	33
Figure 11: Graph of propose method (after inserted timer)	35
Figure 12: Modified Proposition of generated graph	36
Figure 13: Structure of Trace file	40
Figure 14: Figure of TCP connection using NS2	42
Figure 15: Figure of simulated result	50
Figure 16: Figure of Trace file	51
Figure 17: Graph of TCP connection (throughput vs time)	52

Chapter 1

Introduction

The computer network has witnessed tremendous development in recent years, which has resulted in congestion collapse. Congestion reduces performance. TCP's transport layer provides congestion control methods (Transmission Control Protocol)[3]. To control this congestion, TCP employs a variety of methods. "Congestion" simply means "traffic jam". Congestion of automobiles on the road is the same phenomenon that has been employed here. Congestion arises when the flow rate at the sender and recipient is not the same. Congestion can happen in two ways: when network capacity is high, but receiver capacity is low, or when network capacity is low, but receiver capacity is high[3]. When packets arrive at the routers, the routers' function is to deliver the packets. The routers' goal is to send packets to their destinations.

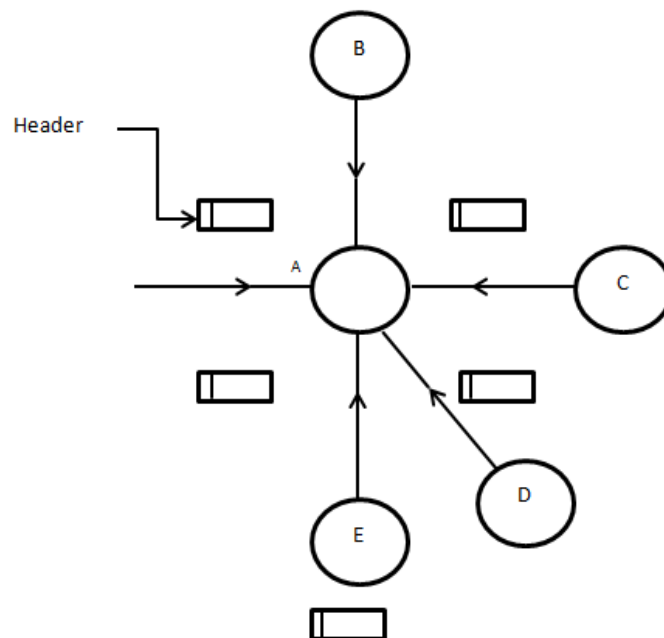


Fig1: Different packet tracks coming to main router.

From the above picture we can see that the packets come to the router from various way. The capability of the router is to peruse the header of the packets and distinguish the way and afterward forward the packets to the particular router. Assume a packet has come to router An and presently router A will conclude whether the packet will go to router B or router C or router D or router or router E. A particular measure of time is expected by the router to peruse the header of the packet

and work out the way. It isn't generally so natural as it appears to be that the router is not a super gadget that every one of the packets that come to the particular router are effortlessly taken care of by the router. A router has its conveying limit and as indicated by that limit it conveys the packets. Be that as it may, on the off chance that an enormous number of packets come to router and it can't convey the packets at similar rate the casings are put away into the cushion of the router. By taking casings from support, the router will convey packets. The support additionally has some restriction and it doesn't have some limitless space. At the point when that space is full then there will happen an information misfortune. In this way, there happens packet misfortune as the support turns out to be full and there will be loss of packets. This present circumstance is called congestion. In reproduction, when the cushion becomes 60-70% full then the presentation of router begins to corrupt. The pace of execution doesn't continue as before.

In past papers, we can see a portion of the calculations that are being utilized. Counting TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms. In the paper, RTT time has been thought of. Full circle time implies when a sign is sent from source to collector and returns to the shipper and for this how much time that is gone on is called round routing time (RRT)[4][5]. In the proper organization this source collector distance will be same, yet the distance will fluctuate in MANET organization. For start to finish transmission key to the convention is the transmission control convention. In this convention the congestion window size is expanded by the update of collectors[4]. On the off chance that the affirmation is effective, it will be accepted that the organization condition is great and congestion window size will be expanded. In this paper it has been shown that the congestion window will be expanded dramatically in sluggish beginning stage and after sluggish beginning stage the congestion window will increment straightly[6]. In any case, in our paper we have attempted to change the sluggish beginning stage where we have embedded a clock from the source's side. Whenever the state of the organization is great, we will send a huge fragment. We will not expand the congestion window dramatically in sluggish beginning stage.

Obviously, congestion results when information traffic surpasses accessible organization assets. Normal supposition that will be that as assets become more reasonable congestion will be dealt with consequently. As per Jain, this leads to the misconception which is, As memory turns out to be more reasonable congestion will be settled consequently since it happens considering a restricted support limit. Congestion will be settled as fast associations open up in light of the fact that it results from slow connections.

Chapter 2

Literature Review

Congestion will be settled naturally as quicker processors arise and are sent since it is brought about by low handling speed[7]. As opposed to these convictions, more congestion and poor execution might be the situation if coordinated endeavors isn't made towards fostering a proper convention plan. High memory limit gadgets and low memory limit gadgets are both helpless against network congestion. For high memory limit gadgets, the bundles are supported and postponed in long lines to such an extent that they break and would have been retransmitted. While, for low memory limit gadgets, abundance traffic start flood and dispose of.

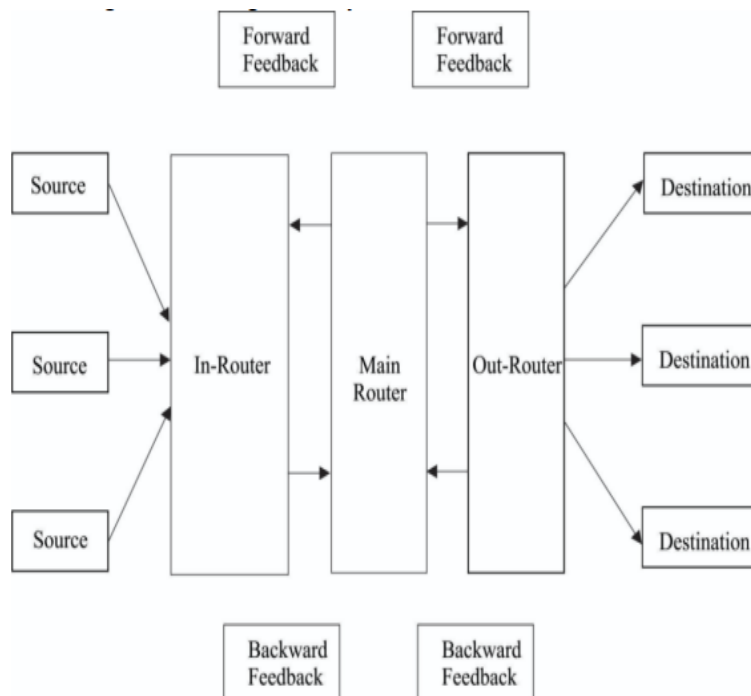


Fig 2. Architecture of the Existing System (Sharadehandra et al. 2017).

Floyd proposed a "condition-based congestion control for unicast applications". As indicated by them, (TCP) has been doing great in overseeing most "best-exertion traffic" in the web today. All things considered, a congestion control procedure that is viable with TCP and tries not to respond to bundle drop by dialing back pace of transmission by half will be a great approach to handling best-exertion unicast streaming interactive media traffic. With this instrument, the source controls

pace of sending relying upon estimation of misfortune occasion. In the meantime, misfortune occasion is what is happening where some parcels are dropped inside a one full circle time. Yang showed that in common organizations (eg. Web), the hosts ought to respond to congestion by changing their pace of transmission in this manner forestalling a breakdown and expanding the accessible organization assets. As indicated by Yang, the web today is powerful because of TCP's host-based congestion control procedures. Much as TCP congestion control performs well for enormous information transmission applications, it wouldn't function admirably for other more up to date applications which will track down TCP conduct of splitting transmission rate considering congestion as unforgiving[3]. As such, since web traffic is for the most part TCP based, it becomes appropriate that arising congestion control procedures be TCP viable. Yang then, at that point, inspected the reasonableness, responsiveness and self-assuredness of TCP, General Additive Increase and Multiplicative Decrease (GAIMD) and some other two run of the mill TCP-Friendly congestion control conventions.

Mawhinney planned a congestion the executive's procedure that smoothen bundles move in an organization. The procedure notices information bundles leaving the organization for congestion signs, and successfully controlling the host applications to free the organization from congestion. The configuration is very like the Congestion Controlling utilizing Network Border convention planned by Sharadchandra then again for the previous, the rate controlling is carried out in the host end and not in the organization. As indicated by Mawhinney, the framework forestall congestion from arriving where information is lost, or parcels are dropped[7].

Modern implementations of TCP contain four intertwined algorithms that have never been fully documented. The purpose of this document is to document these four algorithms for the Internet. Fast retransmit and fast recovery were implemented after RFC 1122, but slow start and congestion avoidance came after that[8]. Slow start adds another window to the sender's TCP: the congestion window. Each time an ACK is received, the congestion window is increased by one segment. This provides an exponential growth, although it is not exactly exponential because the receiver may delay its ACKs. Congestion can occur when data arrives on a big pipe and gets sent out a smaller pipe. Congestion avoidance is a way to deal with lost packets. When congestion occurs TCP must slow down its transmission rate of packets into the network, and then invoke slow start to get things going again. When congestion occurs, one-half of the current window size (the minimum of cwnd and the receiver's advertised window, but at least two segments) is saved in ssthresh. If congestion is indicated by a timeout, cwnd is set to one segment. Modifications to the congestion

avoidance algorithm were proposed in 1990[8]. TCP may generate an immediate acknowledgment when an out-of-order segment is received. If three or more duplicate ACKs are received in a row, it is a strong indication that a segment has been lost. After fast retransmit sends what appears to be the missing segment, congestion avoidance, but not slow start is performed. The reason for not performing slow start in this case is that the receipt of the duplicate ACKs tells TCP more than just a packet has been lost. The fast retransmit algorithm first appeared in the 4.3 BSD Tahoe release, and it was followed by slow start. Security considerations are not discussed in this memo.

A novel RTO timer is designed and implemented for mobile Ad-Hoc network. This paper primarily focuses on TCP timer management in infrastructure less wireless network. Optimizing the congestion window will give the better performance improvement, we have implemented a technique to achieve the same[9]. TCP is the three ways handshaking method for the connection to be established. The sender initiates the request and it sent to the receiver, this can be done in wireless and wired connection. Each and every packet sent by the sender will be waiting for the acknowledgement so that it will start the transmission. The time taken by a signal from a sender to come back to the sender is called the roundtrip time. In the case of wireless acknowledgment can be delayed due to the mobility of the node. When we optimize the RTO timer the retransmission can be avoided and increase performance. The transmission control protocol is developed for infrastructure network. When we use the same protocol for infrastructure-less network then it will not give the better performance as it was purely designed for infrastructure networks[9]. In the beginning a lot of Congestion control algorithms are implemented. Transmission control protocol (TCP) in infrastructure networks can cause performance degradation by means of packet drop. Many reasons for such a big drop in the throughput, in this paper we can see concentrating on the packet drop may be due to acknowledgment due to congestion if suppose. In Mobile Ad-Hoc Network (MANET) round trip time is dynamic. So packet loss occurs if there is no acknowledgment within the stipulated time. Two types of solutions are proposed for the current problem in MANET. By doing this optimization the throughput can improved.

This document defines TCP's four intertwined congestion control algorithms: slow start, congestion avoidance, fast retransmit, and fast recovery. It also describes how TCP should begin transmission after a relatively long idle period, as well as how acknowledgment generation methods should be expressed to the client[10]. This document specifies four TCP congestion

control algorithms. Their use with TCP is standardized. This document also specifies what TCP connections should do after a relatively long idle period. This section defines the four congestion control algorithms: slow start, congestion avoidance, fast retransmit and fast recovery, developed in [Jac88] and [Jac90]. In some situation it may be beneficial for a TCP sender to be more conservative than the algorithms allow. The slow start and congestion avoidance algorithms must be used by a TCP sender to control the amount of outstanding data being injected into the network. A non-standard, experimental TCP extension allows that a TCP MAY use a larger initial window (IW), as defined in equation 1 [AFP98]. Slow start begins with each ACK received that acknowledges new data. During congestion avoidance, cwnd is incremented by 1 full-sized segment per round-trip time (RTT) Equation (2) provides an acceptable approximation to the underlying principle. Older implementations have an additional additive constant on the right-hand side. A TCP receiver SHOULD send an immediate duplicate ACK when an out-of-order segment arrives. Duplicate ACKs can be caused by dropped segments or re-ordering of data segments by the network. The sender should use the "fast retransmit" algorithm to detect and repair loss. Fast retransmit and fast recovery algorithms are usually implemented together. For each additional duplicate ACK received, increment cwnd by SMSS. This artificially inflates the congestion window in order to reflect the additional segment that has left the network. The next ACK should be the acknowledgment elicited by the retransmission from step 1. TCP can potentially send a cwnd-size line-rate burst into the network after an idle period. Slow start serves to restart the ACK clock, just as it does at the beginning of a transfer. Cwnd should be no more than RW before transmission if the TCP has not sent data in an interval exceeding the retransmission timeout[10]. A number of loss recovery algorithms that augment fast retransmit and fast recovery have been suggested by TCP researchers. While some of these algorithms are based on the TCP selective acknowledgment (SACK) option [MMFR96], others do not require SACKs. There are Security Considerations involved. An attacker can cause data packets or their acknowledgments to be lost, or by forging excessive duplicate acknowledgments. Causing two congestion control events back-to-back will often cut ssthresh to its minimum value of $2 \cdot \text{SMSS}$. Conceivably, such an attack could drive a portion of the network into congestion collapse.

The aim of this study is to survey the various modifications of window base congestion control. Mathematical modeling of the internet is discussed and proposals to improve TCP startup were reviewed[1]. Multiple startups for TCP specify that startup speed is selectable from a set of algorithms, we then introduced the e-speed start. Before 1988, TCP was used to ensure that the

Internet could only deliver data not necessarily guaranteeing the delivery. This same TCP version did not include congestion avoidance, fast-recovery and fast-retransmit mechanisms. The direct impact on user applications is low network utility derivation as a result of heavy network congestion. Aims to review existing proposals on TCP congestion avoidance and slow-start state with view to motivating a new direction in the network utility maximization. The paper also looks at the fate of Jacobson's AIMD algorithm and its subsequent modifications in the face of cross-traffic and heterogeneous flows.

TCP Tahoe: Tahoe was the first implementation of TCP to employ congestion control mechanism. Tahoe achieved congestion control by adjusting the size of its window additively to increase and multiplicatively to decrease. Algorithms implemented are slow start, congestion avoidance, fast retransmit and fast recovery.

TCP New Reno: The new-Reno TCP includes a change to the Reno algorithm at the sender end with a view to eliminate Reno's wait for a retransmit time-out when multiple packets are lost from a window. This change modifies the behavior during fast recovery.

SACK (TCP with Selective Acknowledgement): SACK is an extension of TCP Reno and TCP New Reno. It intends to solve two problems of both - detection of multiple packet loss and transmission of more than one lost packet per RTT. SACK retains the slow start and fast retransmits of Reno but also has the coarse-grained time out of Tahoe. The sender exits fast recovery when a acknowledgment is received acknowledging that all outstanding data when fast recovery entered have been received. When a partial ACK is received, the pipe is decreased by two packets rather than one. This limits the number of packets that can be sent in response to a single ACK.

TCP Vegas: Different from other TCP implementations, TCP Vegas avoids the problem of not detecting lost packets when the window is very small and could not receive enough duplicate ACKs. It keeps track of when each packet was sent and calculates an estimate of RTT for each transmission[1]. This is done by monitoring how long it took each ACK to get back to the sender. The internet protocol of choice, TCP New-Reno, has performed well in today's inter-net. How will TCP co-habit with other protocols that are more aggressive and nonresponsive to congestion indication? These are questions that must be answered in finding a replacement for today's internet protocol.

Multipath TCP, as proposed by the IETF working group, allows a single data stream to be split across multiple paths. This has obvious benefits for reliability, and can also lead to more efficient use of networked resources. Our algorithm is a drop-in replacement for TCP, and we believe it is safe to deploy[4]. In this paper, we will restrict our attention to end-to-end mechanisms for sharing capacity, specifically to modifications to TCP's congestion control algorithm. They have demonstrated a working multipath congestion control algorithm. It brings immediate practical benefits: in §5 we saw it seamlessly balance traffic over 3G and WiFi radio links, as signal strength faded in and out. It is safe to use: the fairness mechanism from §2.5 ensures that it does not harm other traffic, and that there is always an incentive to turn it on because its aggregate throughput is at least as good as would be achieved to the best of its available paths. It should be beneficial to the operation of the Internet, since it selects efficient paths and balances congestion, as described in §2.2 and demonstrated in §3, at least in so far as it can give topological constraints and the requirements of fairness. They believe our multipath congestion control algorithm is safe to deploy, either as part of the IETF's efforts to standardize Multipath TCP or with SCTP, and it will perform well. This is timely, as the rise of multipath-capable smartphones and similar devices has made it crucial to find a good way to use multiple interfaces more effectively[4]. Combined with transport protocols such as Multipath TCP or SCTP, or congestion control mechanism avoids the need to make such binary decisions, but instead allows continuous and rapid rebalancing on short timescales as wireless conditions change. The congestion control scheme is designed to be compatible with existing TCP behavior. However, existing TCP has well-known limitations when coping with long high-speed paths. To this end, Microsoft incorporates compound TCP in Vista and Windows 7, although it is not enabled by default, and recent Linux kernels use Cubic TCP. In the paper, they believe that Compound TCP should be a very good match for our congestion control algorithm.

Profiling is an important part of the development of software, but in the past it has been limited to overall performance metrics such as wall clock time. For modern applications, it is necessary to have a programming API that allows the application to query and analyze its own performance characteristics[11]. The Cactus Framework is an open-source, modular, highly portable programming environment for collaborative high-performance computing. Cactus originated in the numerical relativity community but is now used in many disciplines including astrophysics, coastal modeling, and quantum gravity. In the paper, The simulation runs for 1, 737 seconds, spending 19% of the total time check pointing. This functionality can be used to guarantee a certain

level of fault tolerance when adapting the check pointing based on the simulation's characteristics. In another run using the Adapt Check thorn, we rebounded the interval between checkpoints independent of the performance of the 1178 D. Stark et al.run and the I/O system. A scientific code can use a generic, self-contained infrastructure for runtime profiling and adaption leading to significantly improved overall performance. The timing infrastructure was implemented in a highly portable manner in the Cactus Framework. The infrastructure is able to use platform dependent clocks, as well as libraries such as PAPI.

Chapter 3

Methodology

The Transmission Control Protocol (TCP) serves as the Internet's backbone, providing end-to-end congestion control and continuous reliable data delivery services. The TCP congestion control mechanism ensures that data is delivered from start to finish without causing congestion en route to the destination[12][13]. There many type of TCP congestion control algorithm such as TCP reno, TCP vegas, TCP westwood and so on[12].The algorithms which are used in TCP are the reasons we are using Internet comfortably despite the congestion in the network[14]. If we did not use any algorithm, then we would face congestion while using the network. This problem has been by using TCP (Transmission Control Protocol) congestion control algorithms[13].

TCP (Transmission Control Protocol) uses different kinds of algorithms like TCP Tahoe, TCP Reno, TCP New Reno, TCP Vegas,

3.1 TCP Tahoe

The TCP Tahoe was delivered in 1988 by V. Jacobson in being the principal execution of TCP to utilize blockage control component. Tahoe contains the AIMD (added substance increment, augmentation decline) being its control instrument. Tahoe accomplished blockage control through changing its windows size additively to increment and multiplicatively to decrease. AIMD at the underlying stage increments windows size dramatically be that as it may, after a specific limit, it changes to straight window size in-wrinkle for example by one parcel for every RTT before congestion happens (Additive increment). As of now, Tahoe changes to the blockage aversion state. On the off chance that the ACK for a bundle isn't gotten before a break, the sift old set is diminished significantly and the blockage window is reduced to one parcel (Multiplicative lessening)[1]. In aggregate mary, TCP Tahoe controls blockage as follows:

- When clog window is underneath the limit, the blockage window develops dramatically (slow beginning state)
- When the blockage window is over the edge the blockage window develops straightly (added substance in-wrinkle) for example blockage evasion.

- Whenever there is a break, the limit is set to one portion of the ongoing clog window and the blockage window is set to one while the bundle is retransmitted (multiplicative diminishing)
- Algorithms carried out are slow beginning and blockage aversion.

3.2 TCP Reno

Proposition to adjust Tahoe was given in like its ancestor, Reno sets its blockage window to one parcel upon a break (RTO). Be that as it may, Reno stretched out its calculation to incorporate the quick retransmit instrument. The quick retransmit includes the re-transmission of a dropped bundle on the off chance that three copy ACKs for a parcel are gotten before the RTO. Reno likewise presents the quick recuperation instruments which forestall transmission to reemerge the sluggish beginning state after a quick retransmit. Rather the window size is divided and the limit is changed in like manner and TCP stay in blockage evasion until a break happens[12]. This is talked about exhaustively in TCP Reno turned into the standard TCP calculation carried out in many PCs. Calculations carried out by Reno are slow beginning, blockage aversion, quick retransmit and quick recuperation.

3.3 TCP New Reno

The new-Reno TCP incorporates a change to the Reno algorithm at the shipper end so as to dispense with Reno's hang tight for a retransmit break at whatever point different bundles are lost from a window. This change adjusts the shipper's conduct during quick recuperation. At the point when this hap-pens, New Reno doesn't exit from the quick recuperation state as on account of Reno, however hangs tight for the receipt of all the extraordinary ACKS for that window. The followings are the outline of New-Reno quick recuperation activities;

- It takes note of the most extreme parcel s extraordinary while entering quick recuperation
- When another ACK is gotten and it recognizes every one of the exceptional bundles, then, at that point, quick recuperation is left and cwnd is set to around 50% of the worth of ssthresh, then, at that point, it travels to the blockage aversion state. Be that as it may, on the off chance that a fractional ACK is gotten, it expects the following parcel in the connection is lost and attempts to retransmit
- It leaves quick recuperation when all information in the window is recognized.

3.4 TCP Vegas

The TCP blockage control conspires that have been de-scribed up until this point use bundle misfortune base way to deal with measure clog. There is a class of blockage control algorithms that adjust its clog window size in light of start to finish delay. This approach started from and is introduced by as TCP Vegas. The followings are the distinctions between TCP Vegas and TCP Reno:

- In the sluggish beginning state, blockage control was incorporated by a purposeful postpone in clog window development.
- When parcel misfortune happens, TCP Vegas treats the receipt of specific ACKs, as a trigger to look at on the off chance that a period ought to happen.
- It refreshes its blockage window in light of start to finish delay as opposed to involving bundle misfortune as the window update boundary. Vegas broadened Reno re-transmission methodology. It monitors when every bundle was sent and ascertains a gauge of RTT for every transmission. This is finished by observing what amount of time it required for each ACK to return to the shipper. At the point when a copy ACK is gotten, it per-structures the accompanying check: if (current RTT > RTT gauge) If this is valid, it retransmits the parcel without hanging tight for 3 copy ACK or a break as in Reno. Subsequently, Vegas takes care of the issue of not distinguishing lost parcels when the window is tiny for example under three and couldn't get sufficient copy ACKs. TCP Vegas blockage aversion conduct is not quite the same as other TCP executions[1]. It decides block particle states utilizing the sending rate. On the off chance that there is a diminishing in determined pace of transmission because of enormous line in the connection, it decreases its window. While the sending rate expands, the window size additionally increments.

3.5 Delay Based Congestion Control Algorithms

These sorts of calculations use lining postponement to flag the requirement for window change. The issue of reasonableness accompanies these calculations. Postpone based blockage control is appealing in light of the fact that it can take care of the issue of reasonableness utilizing lining delay. To carry out postpone based congestion control, it is important to quantify proliferation delay. Proliferation delay is the time it takes a parcel to make a trip from the shipper to its objective. The proliferation delay is typically set to the littlest noticed RTT.

There are a few noticed issues with the assessment of the lining postponed. Assessing lining delay is challenging in the event that the RTT contains more components, fastened proliferation delay, for example retransmission postpone in remote connection, a high stacked Internet interface and so forth. There are not very many explores done on delay-based blockage control in remote versatile organization except for which proposed postpone based clog control conspire for business CDMA (Code Division Multiple Access). Different lines of explores are in the space of high velocity, enormous postpone networks[1]. Unmistakable among these are the High-Speed TCP (HSTCP) proposed by and scale capable TCP (STCP) proposed by which are experimental conventions that endeavor to further develop TCP performance under enormous data transfer capacity postpone item. They make TCP increases rule become more forceful. Misfortune postpone based Strategy was utilized by TCP Africa, Com-pound TCP and TCP Illinois. These conventions attempt to increment window size more forcefully than TCP New Reno as long as the organization isn't completely used and it changes to AIMD conduct of Reno when the organization is close to blockage.

3.6 Rate-Based Congestion Control Scheme

As indicated by, an extraordinary level of the ebb and flow re-look through on blockage control focus on the utilization of organization utility expansion structure as direction for plan and investigation. The streamlining base structure presented by framed the determined working point for blockage control calculations. The structure, as indicated by partners a utility capability with each stream and expands the total framework utility capability subject to interface limit requirements[1]. This is alluded to as Kelly's framework issue and it is a streamlining issue. Under the rate-based blockage control, clog control plans can be seen as calculations that register the ideal or sub-ideal answers for the Kelly's framework streamlining issue. Blockage control plans can be classified into three: basic, double and primal-dual.

1.Primal Algorithms:

Here the endpoints adjust the source rates progressively founded on the course costs and the connections select a static regulation to decide the connections costs straightforwardly from the appearance rate.

2.Dual Algorithms:

This is an immediate inverse of the basic calculate, the connections adjust the connections costs progressively founded on the connection rates and the end guides select a static regulation toward decide the source rates straightforwardly from the course costs and other source boundaries.

3.Primal-Dual Algorithm:

The calculations in basic family measure blockage utilizing the connections total rate. This elaborates the averaging of criticism from the organization by end focuses (sources). Then again, the calculations in the double family work out the source rate from the course blockage estimates which compares to averaging the source rate before the sources get a criticism of unequivocal clog data. The basic double calculations saw blockage control as decomposable into two sections: Congestion aversion at the source and dynamic line the executives at the connections. Basic double calculations relate rate change with course blockage measure at the source and relate bundle checking likelihood change with interface total rate at the switch. A model is crafted by Liu, where another class of calculations is presented, which is of basic double sort. That is, they highlight dynamic transformations at both the source and the connection closes.

Dependability of basic component under correspondence is broke down by and announced in. Paganini et al proposed a double calculation and showed that it is steady in erratic geographies and postponements. Alpcan and Basar's calculation for basic double was demonstrated to be steady without even a trace of postponement. It was likewise ended up being steady for networks with a solitary bottleneck interface and a few clients when every client might have different RTT.

The algorithms which are used in TCP are the reasons we are using Internet comfortably despite the congestion in the network. If we did not use any algorithm then we would face congestion while using the network. This problem has been by using TCP (Transmission Control Protocol) congestion control algorithms.

We have tried to do a theoretical study of the slow and start algorithm in our paper. For this we have used NS2 to show the simulation of our algorithm. We have also shown the graph of our modified algorithm.

3.7 The TCP Slow-start Algorithm

We have tried to do a theoretical study of the slow and start algorithm in our paper[8]. For this we have used NS2 to show the simulation of our algorithm. To find the solution to this problem we have decided to use congestion avoidance algorithm which is Slow and Start algorithm. TCP (Transmission Control Protocol) uses various kinds of algorithms, but we are using Slow and Start Algorithm. To avoid congestion policy is used by TCP[8][15]. If the network fails deliver the data as the same rate as the sender then the sender must slow down its transmitting rate[1]. Congestion control of TCP has 3 phases:

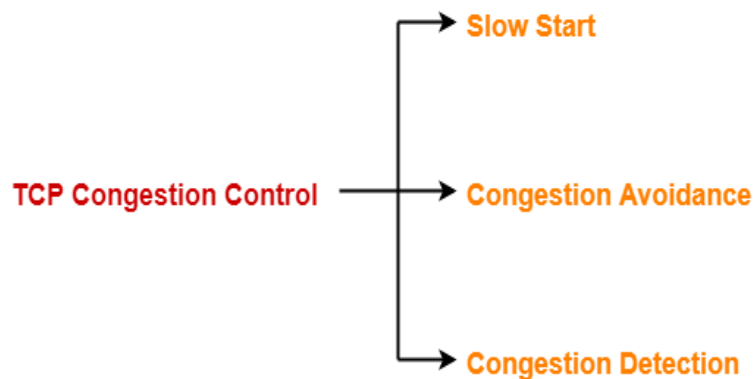


Fig 3: Flow Chart of Slow Start Algorithm

1. Slow Start Phase:

In slow start phase first the sender sends a small segment of data to test whether the receiver can receive it or not. If the sender gets acknowledgement (ACK) before timeout then it means that the sender can send data. Then the sender sends the data exponentially. In this phase the size of congestion window increases exponentially.

The formula is shown as below-

Congestion window size = Congestion window size + Maximum Segment Size (MSS)

This is shown below-

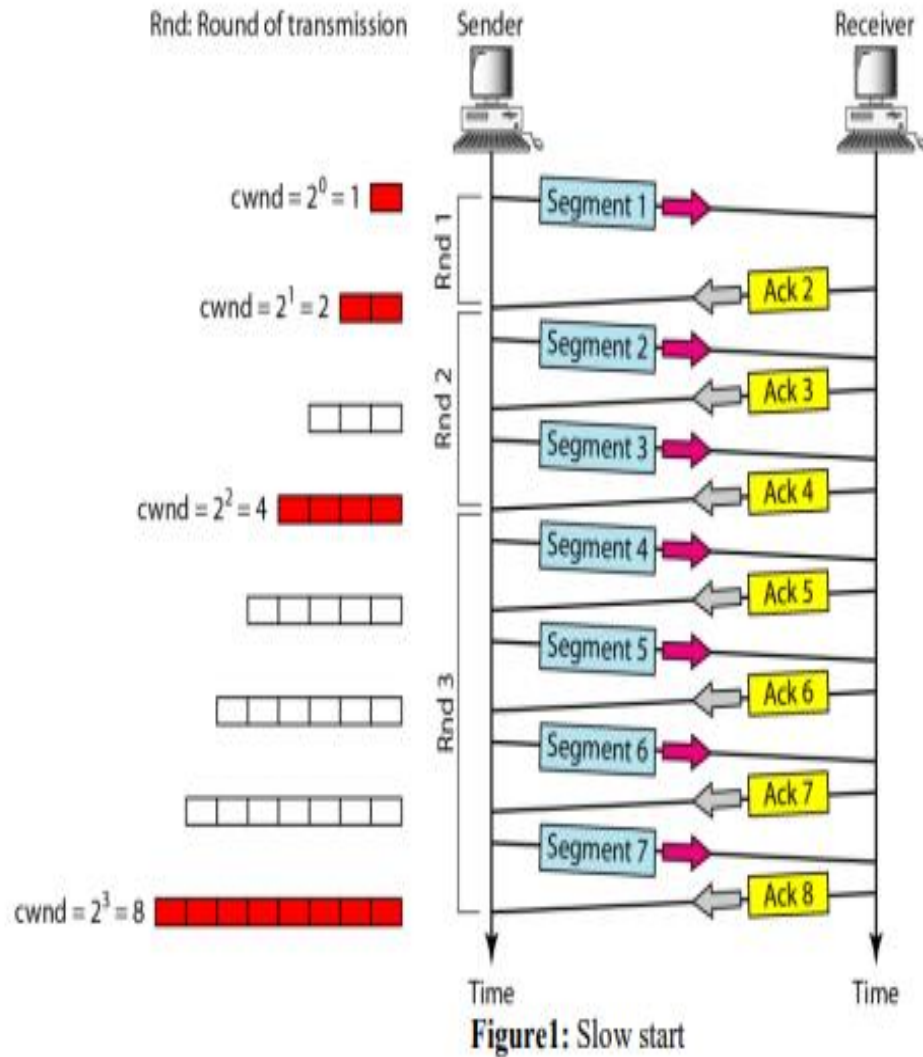


Fig 4: Slow Start Process

Here initially the sender's window size is equal to congestion window size(cwnd) .

The sender starts with initial cwnd size =1 MSS, one segment can be sent initially. After receiving the acknowledgment for segment 1, the cwnd is increased by 1, now cwnd becomes 2. Now two segments can be sent, and these two can also be acknowledged, the cwnd becomes 4. Now 4 segments can be sent when acknowledgement is received cwnd is set to 8.

Initial Congestion Window Size = $(2)^0 = 1\text{MSS}$

Congestion window size after 1RTT (Round Trip Time) = $(2)^1 = 2\text{MSS}$

Congestion window size after 2RTT (Round Trip Time) = $(2)^2 = 4\text{MSS}$

Congestion window size after 3RTT (Round Trip Time) = $(2)^3 = 8\text{MSS}$

This slow start phase continues until the threshold value is reached.

Congestion Avoidance Phase:

After reaching the threshold value congestion avoidance phase starts. The congestion window size is increased linearly by the sender. The congestion window size is incremented to 1 by the sender after receiving each acknowledgement.

The formula is –

Congestion window size = Congestion window size + 1

The congestion avoidance continues until the receiver window size becomes equal to the congestion window size.

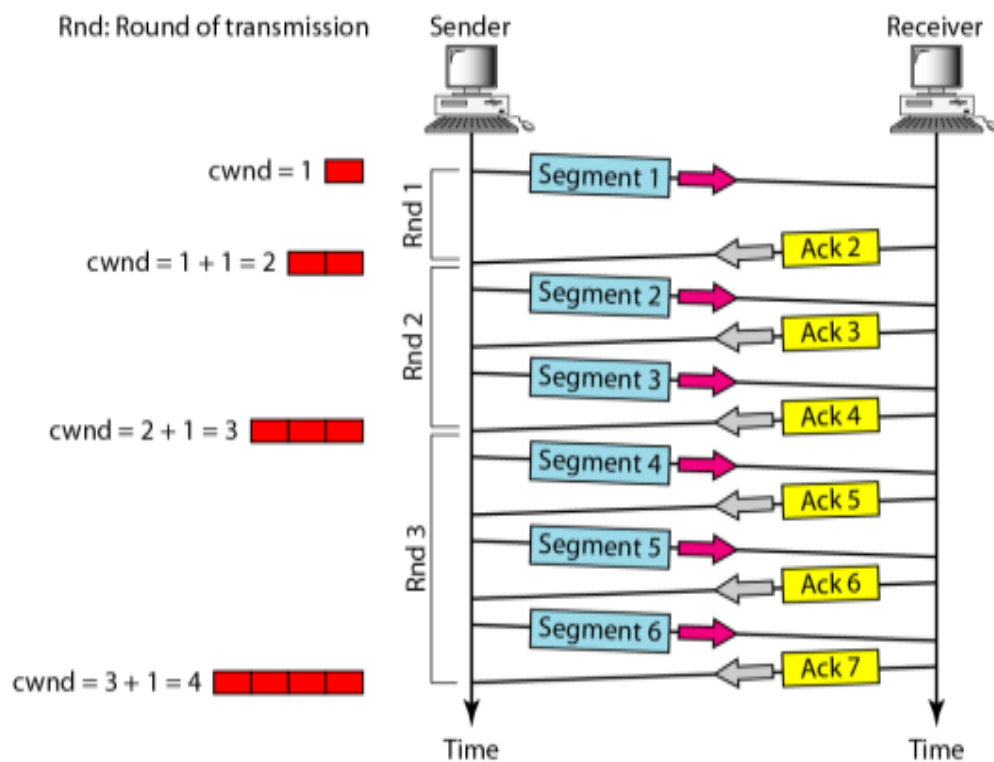


Fig 5: Congestion Avoidance Chart

In slow start algorithm, the size of cwnd increases exponentially. To avoid congestion before it occurs, the exponential growth should slow down. When size of cwnd reaches to the threshold, slow start phase stops. In this algorithm cwnd size will be increment by one.

If we look cwnd in terms of rounds

Start ->cwnd=1

Round 1 ->cwnd=1+1=2

Round 2 ->cwnd=2+1=3

Round 3 ->cwnd=3+1=4

In congestion avoidance phase the size of the cwnd increase additively until congestion is detected.

3.8 Congestion Detection Phase

Whenever sender detects that there has been loss of segments then the sender 2 cases to detect the reason for the loss of segments.

Case 1:

Detection on the time out:

The timer of the time out expires before the acknowledgement is received for a segment. This indicates the congestion in the network strongly. After detecting this case the sender resets the slow start threshold to the half of the current congestion window size. The congestion window size is decreased to 1MSS, and the slow start phase is resumed.

Case 2:

Detection on receiving 3 Duplicate Acknowledgements:

In this case 3 duplicate acknowledgements are received by the sender for a segment. This indicates the weaker probability of congestion in network. It means that when a segment which has been sent was dropped but few segments which were sent later might have reached.

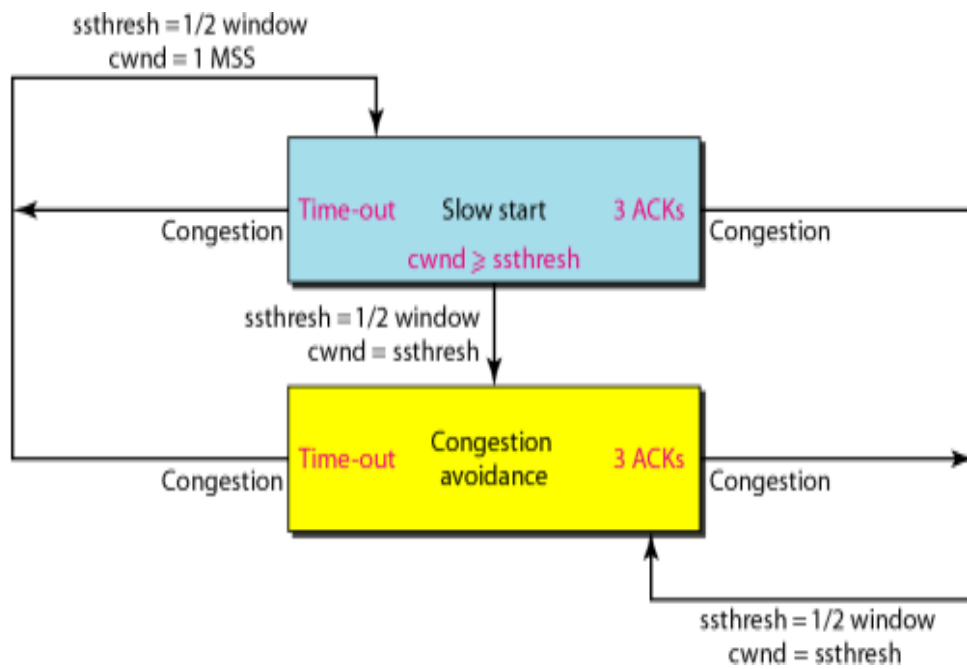


Fig 6: TCP Congestion Policy Chart

Considering a scenario of the algorithm below by depicting a graph,

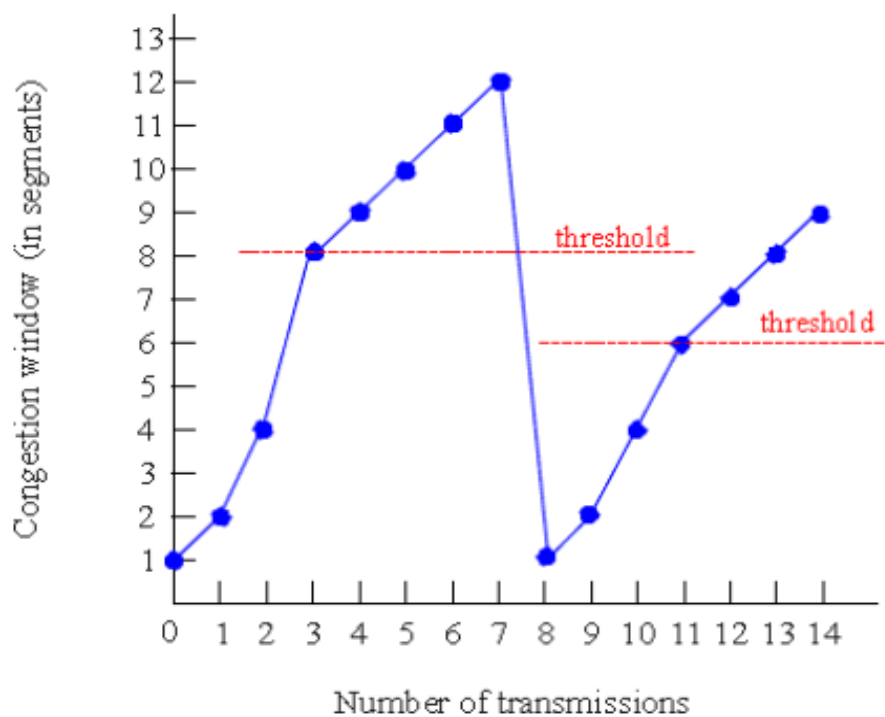


Fig 7: Graph of Transmission x Congestion Window

From the above figure it has been shown that till 2nd transmission the congestion window increases exponentially but at 3rd transmission congestion window touches the threshold value which is 8KB which means that after threshold the network can collapse anytime and congestion window size starts to increase linearly from 4th transmission to 6th transmission to avoid any kind of congestion. After there is a timeout at a point 7 that means congestion has occurred as the acknowledgement did not arrive at the right time. The sender understands that there has been a trouble in the network for which the acknowledgement was not received at the right time. After the timeout at point 7 a threshold will be set which will be half of the timeout point.

At this phase the transmission size will be started from initial congestion window which is 1KB and will start to increase exponentially. When the congestion window touches the new threshold value there is a probability of congestion. That is why the congestion window size increases linearly after the new threshold value. After reaching the timeout value the congestion window size starts from slow start phase.

3.9 Pseudocode

Slow and Start Algorithm :

```

initialize: cwnd = 1
for (each segment ACKed)
    cwnd++;
until (congestion event or cwnd > ssthresh)
/* slowstart is over */
/* cwnd > ssthresh */
every new ACK: cwnd += 1/cwnd
Until (timeout) /* loss event */
If receiving 3ACK for one segment or timeout occurs
    Retransmit the packet
    ssthresh = cwnd / 2
    cwnd = 1
    perform slowstart

```

Flowchart:

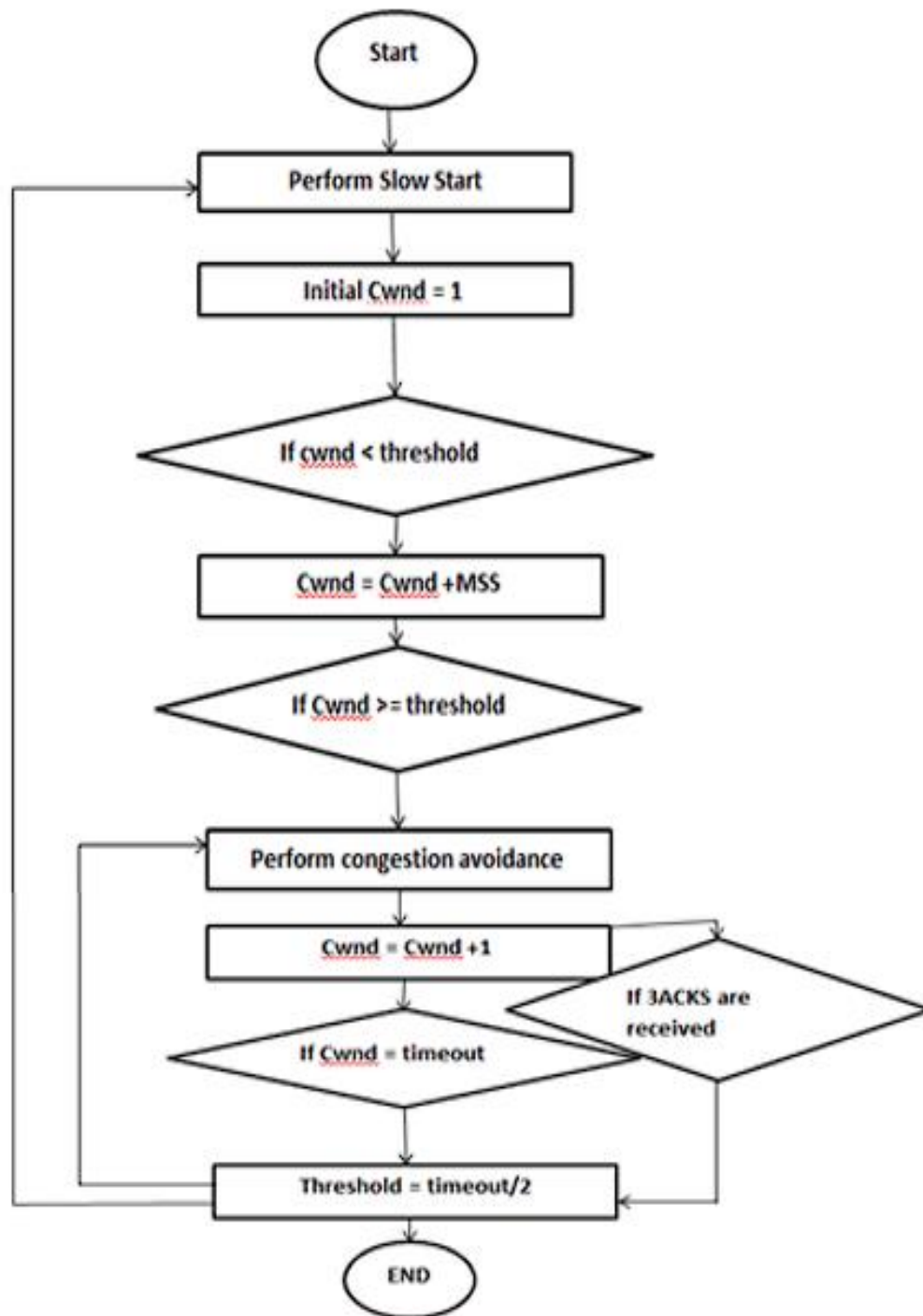


Fig 8: Initial Flow Chart of CCRS

Time of Arrival (TOA)

TOA is characterized as the earliest time at which the sign shows up at the beneficiary. It very well may be estimated by adding the time at which the sign is communicated with the time expected to arrive at the objective (time delay). The time postponement can be registered by splitting the detachment distance between the hubs by the proliferation speed. In TOA, the hubs should be synchronized, and the sign should incorporate the time stamp data. To conquer these limitations, Roundtrip Time of Arrival (RTOA) and Time Difference of Arrival (TDOA) are created.

3.10 The Imbalance Scenery of TCP

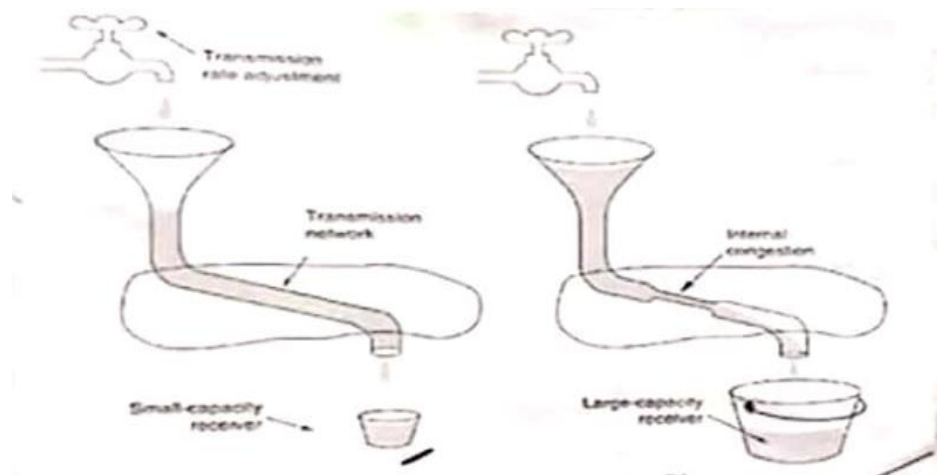


Fig 9: Describing capacity of network when high.

The following photo depicts that the network's capacity is large, while the receiver's capacity is low. On the other side, the receiver's capacity is low, but the network's capacity is low. Both options cause traffic congestion[16].

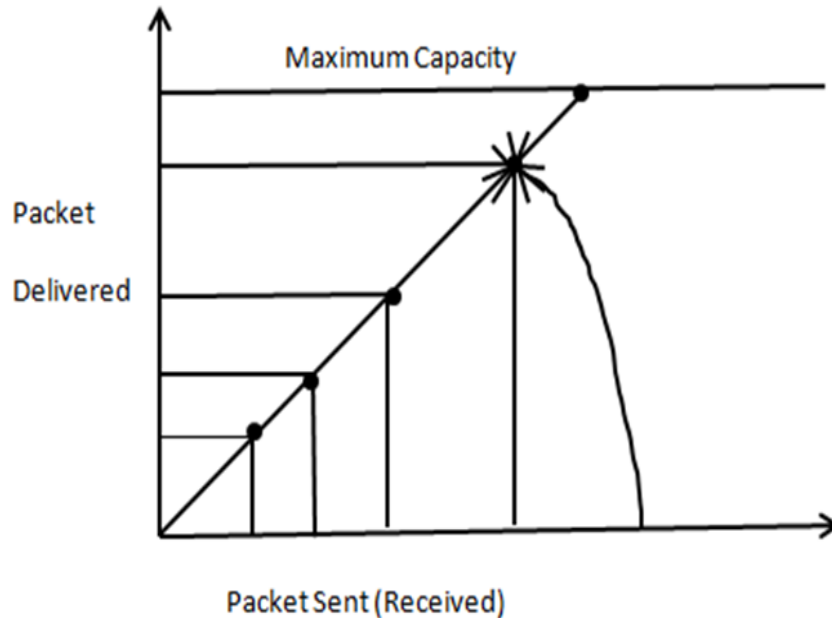


Fig 10: Graph of Packet sent x Packet delivered with respect to maximum capacity.

The above situation illustrates, when a switch is equipped for sending similar amount of information communicated at a similar rate, the diagram will show a straight line. The switch has a most extreme conveyance limit[16]. A switch comprehends the number of bundles a solitary switch that can ship. On the off chance that blockage happens at this stage and is not taken care of, the conveyance limit will before long disintegrate. The switch will not convey by any stretch of the imagination and is close to the same as a dead switch. Since the switch cannot convey any bundles, it becomes seized or closes. On the off chance that the blockage is not controlled sooner rather than later, the switch will come up short. On the off chance that a switch comes up short or dies, it is a gigantic catastrophe for the organization. Subsequently, prior to disintegrating execution, it ought to be controlled. Subsequently, the support cannot be stacked to 60-70 percent limit, and that implies it should be eliminated first. For that reason, controlling blockage is basic.

3.11 The Problem of Slow-Start Algorithm

The main problem of slow and start algorithm is that in the slow start phase we have to send data exponentially though the condition of the network is good, and we can send a large amount of data at a time. When sender's capability and receiver's capability are high that means sender can transmit a large segment and receiver can receive a large segment. But because of using slow and start algorithm sender must send the large segment in small segment. For example, sender needs to send a large of segment of data immediately but if we are using slow and start algorithm then we must send the large segment in a small segment. As a result, a large amount of time is wasted. This is the limitation of this algorithm.

3.12 Our Proposed Method

We have attempted to propose an answer for take care of the issue of slow beginning period of slow and begin calculation. We have added a clock and the congestion window won't increment dramatically[17][9].

For supposition, we have set a clock of 5ms and an edge of 8KB. We, first, will send a segment of 1KB and will sit tight for affirmation. Assuming the affirmation goes inside (0-2) ms and limit is under 8KB then we will expect that the state of the organization is great, and we will send an enormous segment of 4KB. Assuming affirmation goes inside (2-4) ms and limit is under 8KB then we will expect that the state of the organization is great, and we will send an enormous segment of 7KB. Assuming affirmation comes after 5 ms and limit is more prominent than or equivalent to 8KB then we will expect that the state of the organization isn't great and there is a feeling of congestion. In this way, there could drop of parcels out of the blue. As of now we will send a little segment of 1KB. On the off chance that the cwnd comes to break which is 12KB, we will set the edge to the portion of the break which will be 6KB and will begin from slow beginning stage.

Graph:

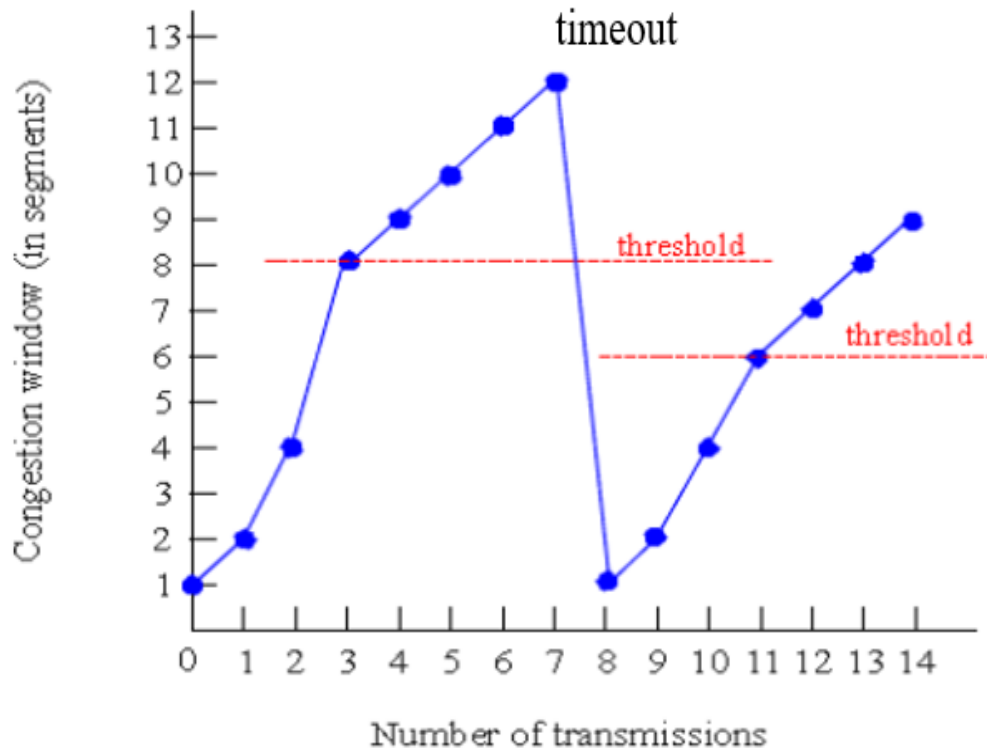


Fig 11: Graph of propose method (after inserted timer)

Algorithm

Suppose the timer is of 10ms and threshold is 64KB.

We have taken the timer is of 5ms and threshold is 8KB.

If((timer == (0-2)ms) && threshold < 8KB)

Cwnd += 4;

If((timer == (2-4)ms) && threshold < 8KB)

Cwnd += 3;

If (timer == 5ms && threshold >= 8KB)

Cwnd += 1;

If (timeout == 12KB)

threshold = timeout/2;

Cwnd = cwnd + 1

Modified Flow Chart with our proposed proposition:

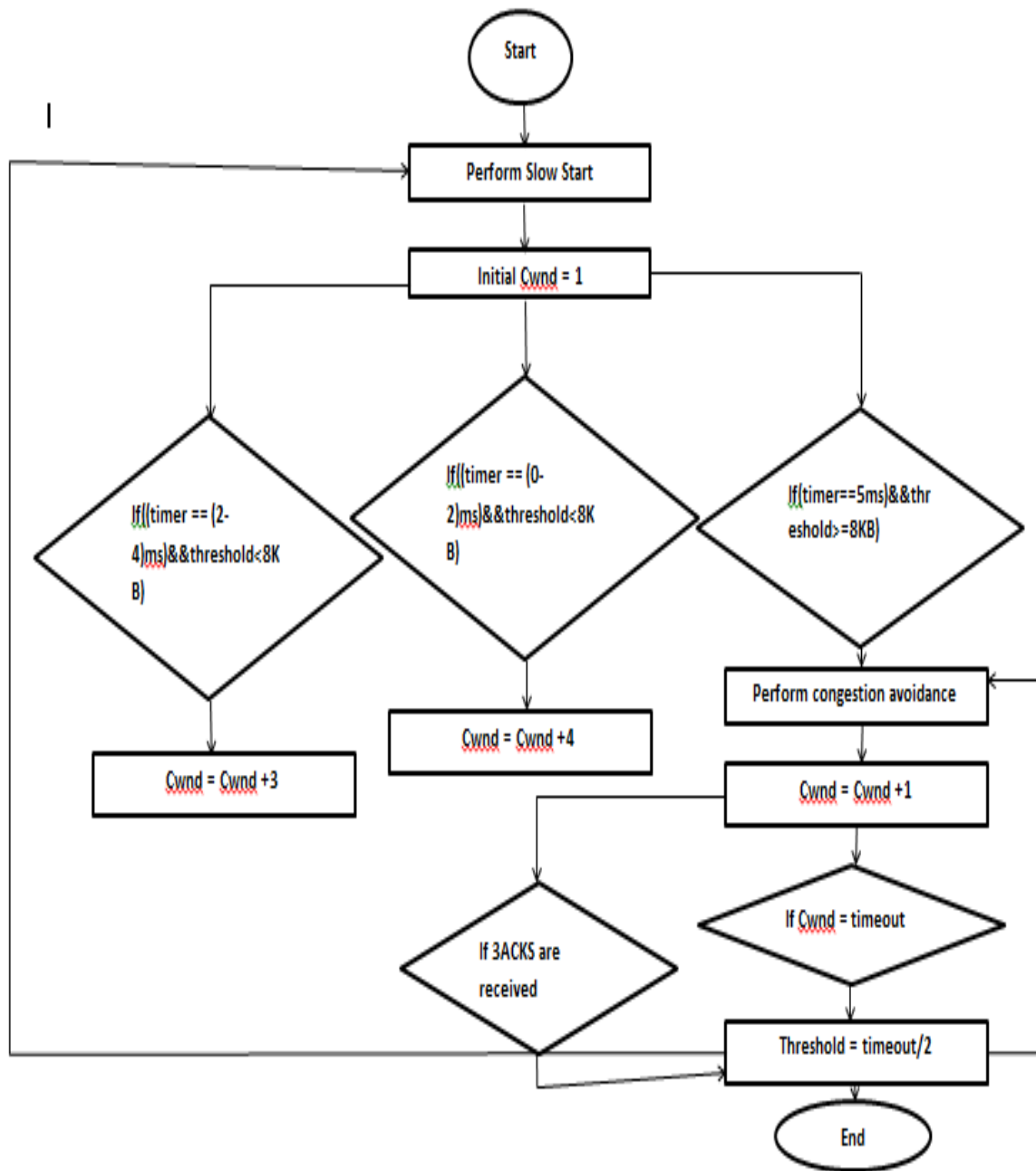


Fig 12: Modified Proposition of generated graph.

Chapter 4

Simulation & Results

4.1 Network Simulator (NS2)

A very wide range of applications, protocols, network types, network components, and traffic models are covered by the ns simulator. The object-oriented C++ simulator and the OTcl (an object-oriented extension of Tcl) interpreter, which is used to run user command scripts, are the two languages on which the NS simulator is built. The collection of network and protocol objects in NS is extensive. Two class hierarchies exist. There is a one-to-one correspondence between the compiled C++ hierarchy and the interpreted OTcl hierarchy[18].

The compiled C++ hierarchy allows us to achieve efficiency in the simulation and faster execution times. This is in particular useful for the detailed definition and operation of protocols. This allows one to reduce packet and event processing time.

Then in the OTcl script provided by the user, we can define a particular network topology, the specific protocols and applications that we wish to simulate (whose behavior is already defined in the compiled hierarchy) and the form of the output that we wish to obtain from the simulator. The OTcl can make use of the objects compiled in C++ through an OTcl linkage (done using tcICL) that creates a matching of OTc object for each of the C++.

In NS, the timing of events controlled by the discrete event simulator and scheduler determines the elapsed time. Pointers and unique IDs to objects that handle events are all characteristics of events that are objects in the C++ hierarchy. The scheduler stores the events to be executed in an ordered data structure (there are four, but by default ns uses a simple linked list), launches each event individually, and activates the event handler. increase.

Millions of people use Tcl (Tool Command Language) worldwide. This language has a fairly straightforward syntax and is very simple to combine with other languages[18]. Rapid development is made possible by the language, which also offers a graphical user interface, compatibility with numerous platforms, flexibility in integration, and is free and simple to use.

The process within NS such as initialization, termination and applications stated below:

An ns simulation starts with the command '*set ns [new Simulator]*' which is thus the first line in the tcl script. In this line, we use the set command to create a new variable called ns, but since it belongs to the simulator class, we can give it any name. The Simulator class was actually instantiated using the new reserved words in the [new Simulator] code. Therefore, you can use all

the methods of the class simulator with this new variable ns. To get an output file that contains data for simulation (trace file) or file for visualization (nam file), you need to create the file using the Open command.

```
#Open the Trace file
set tracefile [open out.tr w]
$ns trace-all $tracefile1
#Open the NAM trace file
set namfile [open out.nam w]
$ns namtrace-all $namfile
```

On top of creating an information trace file named "out.tr" and a reputation visualization trace file for "NAM tools" named "out.nam". inside the TCL script, these files aren't expressly known as by name, however by the pointers declared above, named "tracefile" and "namfile", respectively. Lines one and four of the instance are simply comments, not simulation commands. Note that it starts with a # sign. The second line opens the file to write the "out.tr" declared with the letter "w". The third line uses a machine technique called "trace-all". This method takes as a parameter the name of the file to which the trace is sent[18]. Use this machine command to trace all events within the specific format represented during this chapter. The last line tells the simulator to record all simulation traces in NAM input format.

```
#Define a 'finish' procedure proc finish { } {
global ns tracefile1 namfile
$ns flush-trace close
$t racefile1 close
$namfile
exec nam out.nam &
exit 0
}
```

The word `proc` in this case declares a procedure named `finish` with no arguments. The word `global` is used to indicate that you are using a variable declared outside the procedure. The simulator method `"flush-trace"` outputs the trace to the appropriate file. The tcl `"close"` command closes the previously defined trace file and `exec` runs the `nam` program for visualization. Note that this is an external command, so pass the actual trace filename to `nam` instead of the `namfile` pointer. The `exit` command exits the application and returns the number 0 as the status to the system. The default for clean output is zero. You can use other values to say that this is the end because something went wrong.

Node organization: To create a nodes and links, it has to be pointed by the variable `n0`. When we shall refer to that node in the script, we shall thus write `$n0`. Once we define several nodes, we can define the links that connect them. An example of a definition of a link is: `'$ns duplex-link $n0 $n2 10Mb 10ms DropTail'` which means that nodes `$n0` and `$n2` are connected using a bi-directional link that has 10ms of propagation delay and a capacity of 10 Mb/sec for each direction. To define a directional link instead of a bi-directional one, we should replace `"duplex-link"` by `"simplex-link"`.

Now that you have defined the topology (nodes and links), you need to allow traffic to pass through them. To do this, you must define the routing (namely source and destination), the agent (protocol), and the application that uses them. TCP is a reliable and dynamic congestion control protocol. Use the confirmation generated at the destination to verify that the packet was received successfully. The lost packet is interpreted as a congestion signal. Therefore, TCP requires a two-way connection to confirm going back to the source. There are several variants of the TCP protocol such as Tahoe, Reno, Newreno, and Vegas[12].

The command `'$ns attach-agent $n4 $sink'` defines the destination node. The command `'$ns connect $tcp $sink'` finally makes the TCP connection between the source and destination nodes. The command `'$ns attach-agent $n0 $tcp'` defines the source node of the TCP connection. The command `'set sink [new Agent/TCPSink]'` defines the behavior of the destination node of TCP and assigns to it a pointer called `sink`. In TCP, the destination node has an active role in the acknowledgment protocol to ensure that all packets reach their destination.

Simulation time: NS is a basic discrete event simulation. The Tcl script determines when the event occurs. The NS [new simulator] initializer creates an event scheduler and events are then scheduled using the format:

`$ns at <time> <event>`

The scheduler is started when running ns, i.e. through the command `$ns run`. In our simple example, we should schedule the beginning and end of the FTP applications. This can be done through the following commands:

`$ns at 1.0 "$ftp start"`

`$ns at 20.0 "$ftp stop"`

Therefore, FTP will work from 1.0 until 20.0. Now we are ready to run the entire simulation. If our commands are written to a file named "ex1.tcl", we just need to type "ns ex1.tcl". We have added another procedure at the end that writes the output file with an instantaneous TCP window size at 0.1 second interval. The procedure is recursive, every 0.1 seconds it is called again. It passes as parameters to the TCP source and the file to which we want to write the output. If a random position of the buttons is chosen and unsatisfactory, one can press the "reposition" button and then another random position is chosen[18]. The position can also be changed by clicking the Edit/View button and then "drag" each button to the required position (using the mouse).

Trace file: Tracing NS Simulation objects can generate both visualization traces (for NAM) as well as ASCII file traces corresponding to events recorded on the network. - When we use ns, let's insert four objects into the association: EnqT, DeqT, RecvT and DrpT. EnqT records information about an incoming packet and is queued for the link's input. If the packet overflows, the information about the dropped packet is handled by DrpT. DeqT records information at the time the packet is dropped from the queue. Finally, RecvT gives us information about the packets received when the link was released. NS allows us to receive more information than tracking above. One way is to use queue monitoring.

Event	Time	From node	To node	Pkt type	Pkt size	Flags	Fid	Src addr	Dst addr	Seq num	Pkt id
-------	------	-----------	---------	----------	----------	-------	-----	----------	----------	---------	--------

Figure 13: Structure of trace files

In this above figure we can see the structure of trace files, where in the first place the event status is shown. It can have the values “r”, “+”, “-”, “d” for “received”, “queue”, “dequeue”, and “dropped” and “r”, “s”, “f”, “D” for “received”, “sent”, “forward”, and “dropped” respectively. Then the second place is the time of the event occurs. For the third and forth place is to indicate the source and destination node. Then packet type which can be tcp, awk or udp etc. Then the packet size is placed. After this comes the IP source and destination addresses, flags.

AWK file: NS simulator handles data files with awk. The awk utility allows us to perform simple operations on the data file such as averaging the values of a certain column, adding or multiplying terms by terms between multiple columns, all formatting tasks. data back, etc.

Gnuplot: Gnuplot is widely available freeware for Unix/Linux and Windows operating systems. It has a help command that can be used to find out the details of its operation. The easiest way to use Gnuplot is to type "plot ", where the file & # 40; the name we write fn & # 41; there are two columns representing the x and y values of the points.

Xgraph: Xgraph is a graphing utility provided by ns. Sometimes you have to compile separately with config then create in xgraph folder. Also, sometimes it doesn't work with xgraph which comes with the whole single ns package, and then it can be downloaded and installed separately). Note that it allows you to create replays, Tgif files and other files, by clicking the "Hdcp" button. It can be billed in the TCL command, thus resulting in immediate display after the end of the simulation. As input, the xgraph command requires one or more ASCII files containing each pair of 2-y data points per line.

NS Simulator can provide a lot of detailed data about events happening on the network. If we want to analyze data, we may need to extract relevant information from the traces and manipulate them. Of course one can write programs in any programming language capable of managing data files. However, some tools that seem particularly suitable for these purposes already exist and are freely available on various operating systems (Unix, Linux, Windows, etc.). They only require writing short pieces of code that are interpreted and executed without compiling[18].

4.2 Simulation

Here we have developed a basic structure of TCP connection using ns2. While connecting the nodes with each other, we give them different bandwidth and delay so that we get a better understanding of congestion in between the connected nodes. And inserted FTP between the source to destination. FTP is used for file transfer in a TCP network connection. With the help of NAM which is a TCL/TK based animation tool for showing network simulation traces and real-world packet traces.

The basic structure of TCP connection is shown below:

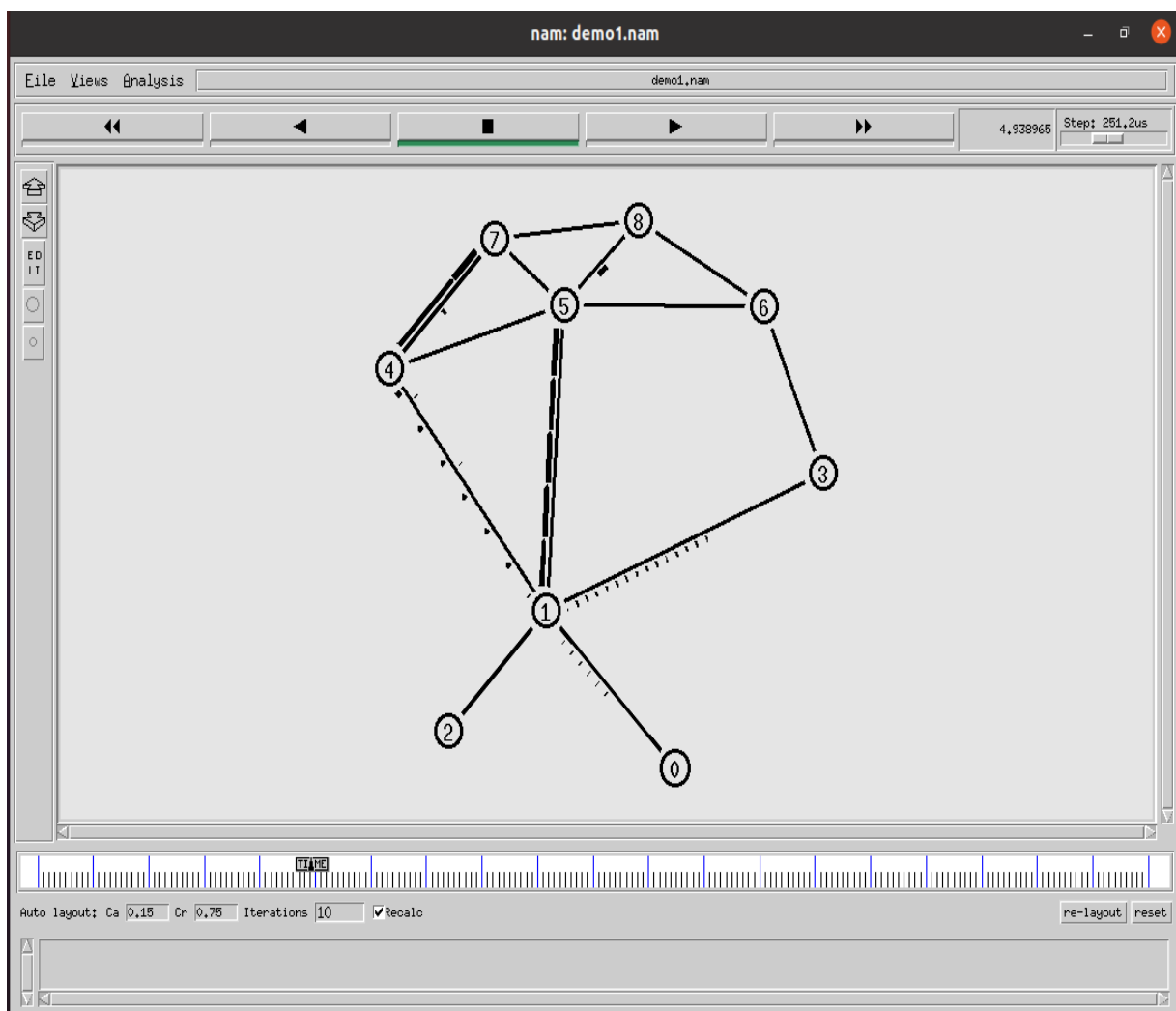


Fig 14: Figure of TCP connection using NS2

From the above figure13 we can see that the nodes \$n1 and \$n0 are connected using a bi-directional link that has 10ms of propagation delay and a capacity of 100 Mb/sec for each direction and the queue size is 50. The nodes \$n3 and \$n1 are connected using a bi-directional link that has 10ms of propagation delay and a capacity of 100 Mb/sec for each direction and the queue size is 50. The nodes \$n4 and \$n5 are connected using a bi-directional link that has 10ms of propagation delay and a capacity of 100 Mb/sec for each direction and the queue size is 50. The nodes \$n8 and \$n5 are connected using a bi-directional link that has 10ms of propagation delay and a capacity of 100 Mb/sec for each direction and the queue size is 50. The nodes \$n2 and \$n1 are connected using a bi-directional link that has 5ms of propagation delay and a capacity of 60 Mb/sec for each direction and the queue size is 50. The nodes \$n1 and \$n4 are connected using a bi-directional link that has 5ms of propagation delay and a capacity of 60 Mb/sec for each direction and the queue size is 50. The nodes \$n1 and \$n5 are connected using a bi-directional link that has 2ms of propagation delay and a capacity of 20 Mb/sec for each direction and the queue size is 20. the nodes \$n6 and \$n5 are connected using a bi-directional link that has 2ms of propagation delay and a capacity of 20 Mb/sec for each direction and the queue size is 20. The nodes \$n5 and \$n7 are connected using a bi-directional link that has 1ms of propagation delay and a capacity of 10 Mb/sec for each direction and the queue size is 5. The nodes \$n3 and \$n6 are connected using a bi-directional link that has 1ms of propagation delay and a capacity of 10 Mb/sec for each direction and the queue size is 5. The nodes \$n6 and \$n8 are connected using a bi-directional link that has 1ms of propagation delay and a capacity of 10 Mb/sec for each direction and the queue size is 5. The nodes \$n4 and \$n7 are connected using a bi-directional link that has 1ms of propagation delay and a capacity of 10 Mb/sec for each direction and the queue size is 5. The nodes \$n7 and \$n8 are connected using a bi-directional link that has 1ms of propagation delay and a capacity of 10 Mb/sec for each direction and the queue size is 5.

After connecting all the nodes in tcl script, we opened the terminal and command ‘ns demo1.tcl’ then the NAM file run to visualize the assemble. To modify the assemble of the node we can use the re-layout and reset button. And then we can run the NAM file and see the packet transformation from one node to another.

4.3 Code of the Proposed Method

```
set ns [new Simulator]
set nf [open demo1.nam w]
$ns namtrace-all $nf

set nt [open demo1.tr w]
$ns trace-all $nt

proc finish {} {
    global ns nf nt
    $ns flush-trace
    close $nf
    close $nt
    exec nam demo1.nam &
    exec awk -f demo1-throughput.awk demo1.tr &
    exec awk -f demo1_packet.awk demo1.tr &
    exit 0
}

#nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]
set n7 [$ns node]
set n8 [$ns node]
```

#connectionss

\$ns duplex-link \$n1 \$n0 100Mb 10ms DropTail

\$ns queue-limit \$n1 \$n0 50

\$ns duplex-link \$n3 \$n1 100Mb 10ms DropTail

\$ns queue-limit \$n3 \$n1 50

\$ns duplex-link \$n4 \$n5 100Mb 10ms DropTail

\$ns queue-limit \$n4 \$n5 50

\$ns duplex-link \$n8 \$n5 100Mb 10ms DropTail

\$ns queue-limit \$n8 \$n5 50

\$ns duplex-link \$n2 \$n1 60Mb 5ms DropTail

\$ns queue-limit \$n2 \$n1 50

\$ns duplex-link \$n1 \$n4 60Mb 5ms DropTail

\$ns queue-limit \$n1 \$n4 50

\$ns duplex-link \$n1 \$n5 20Mb 2ms DropTail

\$ns queue-limit \$n1 \$n5 20

\$ns duplex-link \$n6 \$n5 20Mb 2ms DropTail

\$ns queue-limit \$n6 \$n5 20

\$ns duplex-link \$n5 \$n7 10Mb 1ms DropTail

\$ns queue-limit \$n5 \$n7 5

\$ns duplex-link \$n3 \$n6 10Mb 1ms DropTail

\$ns queue-limit \$n3 \$n6 5

\$ns duplex-link \$n6 \$n8 10Mb 1ms DropTail

\$ns queue-limit \$n6 \$n8 5

\$ns duplex-link \$n4 \$n7 10Mb 1ms DropTail

\$ns queue-limit \$n4 \$n7 5

\$ns duplex-link \$n7 \$n8 10Mb 1ms DropTail

\$ns queue-limit \$n7 \$n8 5

#agents

set tcp1 [new Agent/TCP]

\$tcp1 set class_ 1

\$ns attach-agent \$n0 \$tcp1

set sink1 [new Agent/TCPSink]

```
$ns attach-agent $n7 $sink1
$ns connect $tcp1 $sink1
```

```
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ftp1 set type_ FTP
$ftp1 set packet_size_ 20
$ftp1 set rate_ 8kb
```

```
set tcp2 [new Agent/TCP]
$tcp2 set class_ 2
$ns attach-agent $n8 $tcp2
set sink2 [new Agent/TCPSink]
$ns attach-agent $n5 $sink2
$ns connect $tcp2 $sink2
```

```
set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2
$ftp2 set type_ FTP
$ftp2 set packet_size_ 20
$ftp2 set rate_ 8kb
```

```
set tcp3 [new Agent/TCP]
$tcp3 set class_ 3
$ns attach-agent $n3 $tcp3
set sink3 [new Agent/TCPSink]
$ns attach-agent $n5 $sink3
$ns connect $tcp3 $sink3
```

```
set ftp3 [new Application/FTP]
$ftp3 attach-agent $tcp3
$ftp3 set type_ FTP
```

```
$ftp3 set packet_size_ 20
```

```
$ftp3 set rate_ 8kb
```

```
#start-finish
```

```
$ns at 1.0 "$ftp1 start"
```

```
$ns at 2.0 "$ftp2 start"
```

```
$ns at 2.5 "$ftp3 start"
```

```
$ns at 20.0 "$ftp1 stop"
```

```
$ns at 20.0 "$ftp2 stop"
```

```
$ns at 20.0 "$ftp3 stop"
```

```
proc plotWindow {tcpSource filename} {
```

```
    global ns
```

```
    set now [$ns now]
```

```
    set cwnd [$tcpSource set cwnd_]
```

```
    puts $filename "$now $cwnd"
```

```
    $ns at [expr $now+0.1] "plotWindow $tcpSource $filename"
```

```
}
```

```
set outfile1 [open "tcp1.plot" w]
```

```
$ns at 0.0 "plotWindow $tcp1 $outfile1"
```

```
set outfile2 [open "tcp2.plot" w]
```

```
$ns at 0.0 "plotWindow $tcp2 $outfile2"
```

```
set outfile3 [open "tcp3.plot" w]
```

```
$ns at 0.0 "plotWindow $tcp3 $outfile3"
```

```
$ns at 25.0 "finish"
```

```
$ns run
```

4.4 AWK file of total packet calculation

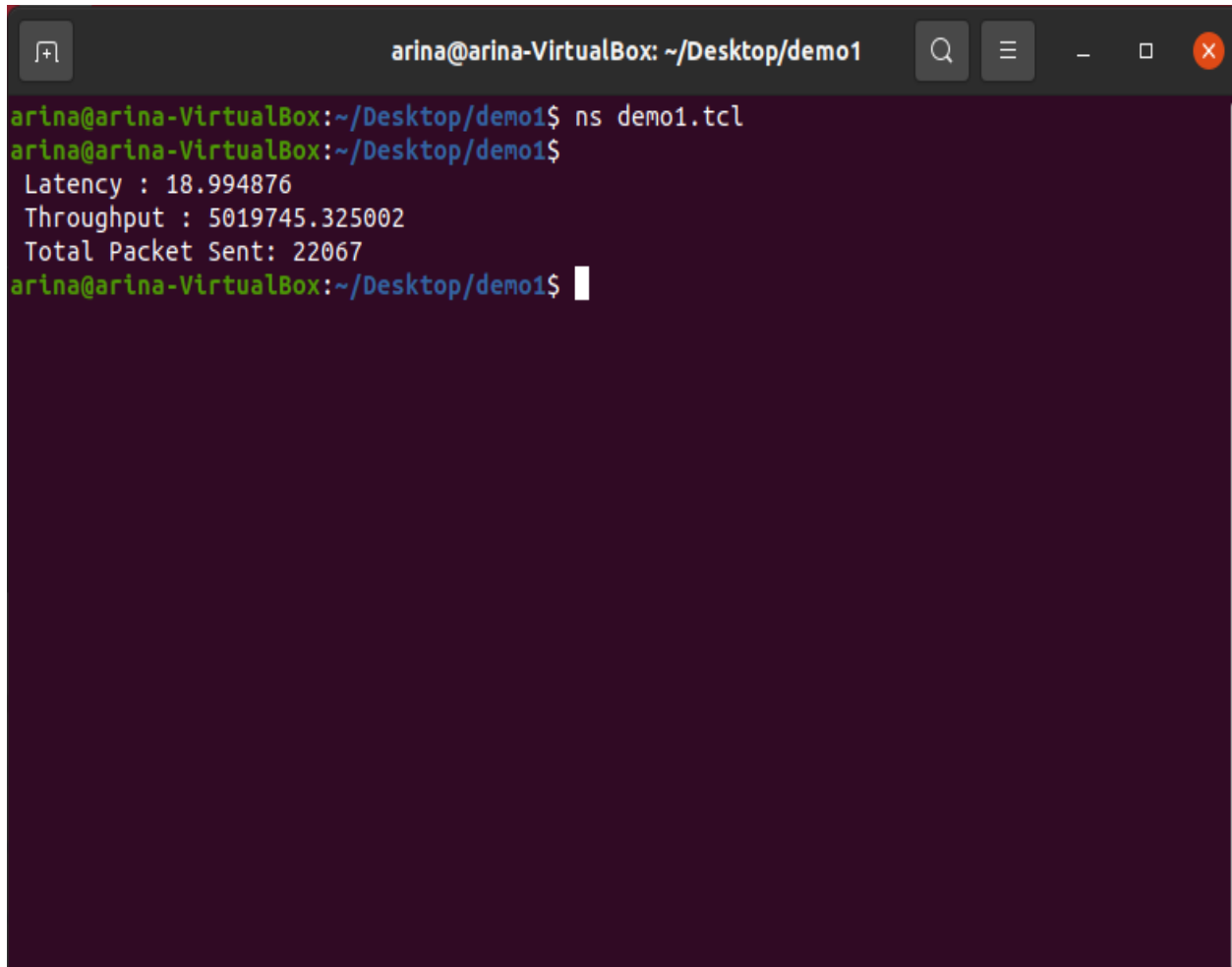
```
BEGIN{
receive=0
drop=0
total=0
ratio=0
}
{
if($1=="r" && $4==4)
{
receive++
}
if($1=="d" && $4==4)
{
drop++
}
}
END{
total=receive+drop
ratio=(receive/total)*100
printf("\n Total Packet Sent: %d", total)
printf("\n Packet Delivery Ratio: %f", ratio)
}
```


4.5 AWK file of Throughput and Latency Calculation

```
BEGIN{
start_time=0
finish_time=0
flag=0
file_size=0
throughput=0
latency=0
}
{
if($1=="r"&&$4==4)
{
file_size+=$6
if(flag==0)
{
start_time=$2
flag=1
}
finish_time=$2
}
}
END{
latency=finish_time-start_time
throughput=(file_size*8)/latency
printf("\n Latency : %f",latency)
printf("\n Throughput : %f",throughput)
```

4.6 Simulated Result

Here is the results of throughput, total packet and latency

A terminal window titled 'arina@arina-VirtualBox: ~/Desktop/demo1' with standard window controls. The terminal output shows the execution of 'ns demo1.tcl' and the resulting simulation metrics.

```
arina@arina-VirtualBox:~/Desktop/demo1$ ns demo1.tcl
arina@arina-VirtualBox:~/Desktop/demo1$
Latency : 18.994876
Throughput : 5019745.325002
Total Packet Sent: 22067
arina@arina-VirtualBox:~/Desktop/demo1$
```

Fig 15: Figure of simulated result

4.7 Trace file

This is the trace file of our tcp connection. As we can see from the first line the “+” represent that the packet is inserted the queue, then the event occur time, then the source and destination node in n1 and n0, then packet type is tcp and then the flag, IP addresses of the source and destination node. Trace file helps to calculate and plot a graph based on the total packet transfer and the event time as it contains all the information of the transmission.

```
1 + 1 0 1 tcp 40 ----- 1 0.0 7.0 0 0
2 - 1 0 1 tcp 40 ----- 1 0.0 7.0 0 0
3 r 1.010003 0 1 tcp 40 ----- 1 0.0 7.0 0 0
4 + 1.010003 1 4 tcp 40 ----- 1 0.0 7.0 0 0
5 - 1.010003 1 4 tcp 40 ----- 1 0.0 7.0 0 0
6 r 1.015009 1 4 tcp 40 ----- 1 0.0 7.0 0 0
7 + 1.015009 4 7 tcp 40 ----- 1 0.0 7.0 0 0
8 - 1.015009 4 7 tcp 40 ----- 1 0.0 7.0 0 0
9 r 1.016041 4 7 tcp 40 ----- 1 0.0 7.0 0 0
10 + 1.016041 7 4 ack 40 ----- 1 7.0 0.0 0 1
11 - 1.016041 7 4 ack 40 ----- 1 7.0 0.0 0 1
12 r 1.017073 7 4 ack 40 ----- 1 7.0 0.0 0 1
13 + 1.017073 4 1 ack 40 ----- 1 7.0 0.0 0 1
14 - 1.017073 4 1 ack 40 ----- 1 7.0 0.0 0 1
15 r 1.022078 4 1 ack 40 ----- 1 7.0 0.0 0 1
16 + 1.022078 1 0 ack 40 ----- 1 7.0 0.0 0 1
17 - 1.022078 1 0 ack 40 ----- 1 7.0 0.0 0 1
18 r 1.032081 1 0 ack 40 ----- 1 7.0 0.0 0 1
19 + 1.032081 0 1 tcp 1040 ----- 1 0.0 7.0 1 2
20 - 1.032081 0 1 tcp 1040 ----- 1 0.0 7.0 1 2
21 + 1.032081 0 1 tcp 1040 ----- 1 0.0 7.0 2 3
22 - 1.032164 0 1 tcp 1040 ----- 1 0.0 7.0 2 3
23 r 1.042164 0 1 tcp 1040 ----- 1 0.0 7.0 1 2
24 + 1.042164 1 4 tcp 1040 ----- 1 0.0 7.0 1 2
25 - 1.042164 1 4 tcp 1040 ----- 1 0.0 7.0 1 2
26 r 1.042247 0 1 tcp 1040 ----- 1 0.0 7.0 2 3
27 + 1.042247 1 4 tcp 1040 ----- 1 0.0 7.0 2 3
28 - 1.042303 1 4 tcp 1040 ----- 1 0.0 7.0 2 3
29 r 1.047303 1 4 tcp 1040 ----- 1 0.0 7.0 1 2
30 + 1.047303 4 7 tcp 1040 ----- 1 0.0 7.0 1 2
31 - 1.047303 4 7 tcp 1040 ----- 1 0.0 7.0 1 2
32 r 1.047442 1 4 tcp 1040 ----- 1 0.0 7.0 2 3
33 + 1.047442 4 7 tcp 1040 ----- 1 0.0 7.0 2 3
34 - 1.048135 4 7 tcp 1040 ----- 1 0.0 7.0 2 3
35 r 1.049135 4 7 tcp 1040 ----- 1 0.0 7.0 1 2
36 + 1.049135 7 4 ack 40 ----- 1 7.0 0.0 1 4
37 - 1.049135 7 4 ack 40 ----- 1 7.0 0.0 1 4
38 r 1.049967 4 7 tcp 1040 ----- 1 0.0 7.0 2 3
39 + 1.049967 7 4 ack 40 ----- 1 7.0 0.0 2 5
40 - 1.049967 7 4 ack 40 ----- 1 7.0 0.0 2 5
41 r 1.050167 7 4 ack 40 ----- 1 7.0 0.0 1 4
42 + 1.050167 4 1 ack 40 ----- 1 7.0 0.0 1 4
43 - 1.050167 4 1 ack 40 ----- 1 7.0 0.0 1 4
44 r 1.050999 7 4 ack 40 ----- 1 7.0 0.0 2 5
45 + 1.050999 4 1 ack 40 ----- 1 7.0 0.0 2 5
46 - 1.050999 4 1 ack 40 ----- 1 7.0 0.0 2 5
47 r 1.055172 4 1 ack 40 ----- 1 7.0 0.0 1 4
48 + 1.055172 1 0 ack 40 ----- 1 7.0 0.0 1 4
49 - 1.055172 1 0 ack 40 ----- 1 7.0 0.0 1 4
50 r 1.056004 4 1 ack 40 ----- 1 7.0 0.0 2 5
```

Fig 16: Figure of Trace file

4.8 Graph

This is the three TCP connection graph with different bandwidth and delay, where x-axis represents the time and y-axis represents the throughput. For every 0.1 sec it is counting the cwnd. At 20ms congestion is detected for the three connection and packet drop is occurred.

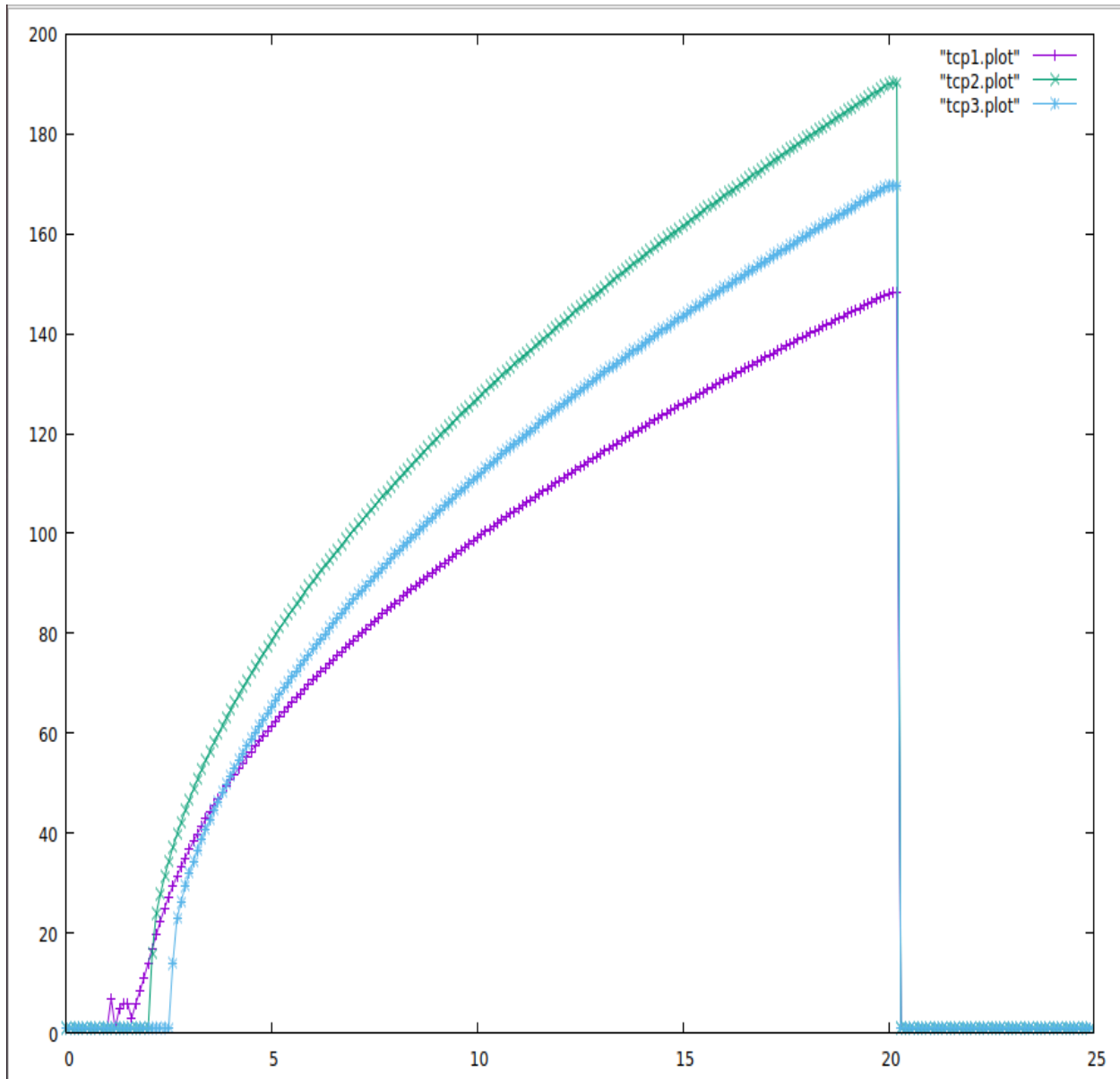


Fig 17: Graph of TCP connection (throughput vs time)

Chapter 5

Conclusion and Future Work

In this paper we have concentrated sluggish and begin calculation hypothetically and we have showed the reproduction of the calculation utilizing NS2(Network Simulator). The superior variant can undoubtedly manage the congestion at whatever point there is a feeling of congestion, and the data transfer capacity isn't enough for the organization, the parcels begin to drop.

We didn't get the enough opportunity to gauge the exhibition of our superior calculation. So, we have chosen if we at any point find the opportunity we will attempt to do as such.

Chapter 6

References

- [1] K. I. Oyeyinka, A. O. Oluwatope, A. T. Akinwale, O. Folorunso, G. A. Aderounmu, and O. O. Abiona, “TCP Window Based Congestion Control -Slow-Start Approach,” *Commun. Netw.*, vol. 03, no. 02, pp. 85–98, 2011, doi: 10.4236/cn.2011.32011.
- [2] X. Wenxian, L. Zhen, G. Guohong, and W. Wenlong, “An Application of the Modification of Slow Start Algorithm in Campus Network,” *Energy Procedia*, vol. 17, pp. 1326–1331, 2012, doi: 10.1016/j.egypro.2012.02.247.
- [3] J. Nagle, “Congestion control in IP,” vol. 14, no. 4, pp. 61–65, 1984.
- [4] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, “Design, implementation and evaluation of congestion control for multipath TCP,” *Proc. NSDI 2011 8th USENIX Symp. Networked Syst. Des. Implement.*, no. March, pp. 99–112, 2011.
- [5] W. (1997) A. A. of F. E. in the C. I. 1997 [online] A. <http://www.cdc.gov/elcosh/docs/d0600/d000645/d000645.htm>. Schriver, “No Title اثر التطور في تكنولوجيا المعلومات,” p. 11, 2001.
- [6] Z. Zhu and W. Liang, “Congestion Control Strategy of Distance Education System Based on TCP Protocol,” *Proc. - 2019 Int. Conf. Intell. Transp. Big Data Smart City, ICITBS 2019*, pp. 485–488, 2019, doi: 10.1109/ICITBS.2019.00123.
- [7] S. Park and H. Park, “Combined oversampling and undersampling method based on slow-start algorithm for imbalanced network traffic,” *Computing*, vol. 103, no. 3, pp. 401–424, 2021, doi: 10.1007/s00607-020-00854-1.
- [8] M. Allman, V. Paxson, and W. Stevens, “RFC 2581: TCP Congestion Control,” *Req. Comments*, pp. 1–14, 1999.
- [9] S. Periyasamy and J. A. J. M, “TCP Congestion Window and Retransmission Timer Optimization in Mobile AdHOC Network,” no. June, 2015.
- [10] TCP, “Congestion control”.
- [11] D. Stark, G. Allen, T. Goodale, T. Radke, and E. Schnetter, “An extensible timing infrastructure for adaptive large-scale applications,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 4967 LNCS, pp. 1170–1179, 2008, doi: 10.1007/978-3-540-68111-3_124.

- [12] A. Macura, E. Missoni, and Z. Kordic, "Comparison of Westwood, New Reno and Vegas TCP congestion control," *23rd DAAAM Int. Symp. Intell. Manuf. Autom. 2012*, vol. 1, no. 2, pp. 151–154, 2012, doi: 10.2507/23rd.daaam.proceedings.035.
- [13] B. Francis, V. Narasimhan, and A. Nayak, "Enhancing TCP Congestion Control for Improved," pp. 472–483.
- [14] X. Zhang and A. Papachristodoulou, "Improving the performance of network congestion control algorithms," *IEEE Trans. Automat. Contr.*, vol. 60, no. 2, pp. 522–527, 2015, doi: 10.1109/TAC.2014.2336338.
- [15] M. Marchese, "Modifications of the slow start algorithm to improve tcp performance over large delay satellite channels," *IETE J. Res.*, vol. 52, no. 2–3, pp. 121–137, 2006, doi: 10.1080/03772063.2006.11416449.
- [16] A. Mudassar, N. Md Asri, A. Usman, K. Amjad, I. Ghafir, and M. Arioua, "A new Linux based TCP congestion control mechanism for long distance high bandwidth sustainable smart cities," *Sustain. Cities Soc.*, vol. 37, pp. 164–177, 2018, doi: 10.1016/j.scs.2017.11.005.
- [17] نقش منابع اطلاعاتی مورد استفاده بیماران دیابتی در مدیریت بیماری، "ف. ص. ل. ا. ر. ا. ا. صباحی، 59. آنها"، vol. 59.
- [18] F. G. Becker *et al.*, "No 主観的健康感を中心とした在宅高齢者における 健康関連指標に関する共分散構造分析Title," *Syria Stud.*, vol. 7, no. 1, pp. 37–72, 2015, [Online]. Available: https://www.researchgate.net/publication/269107473_What_is_governance/link/548173090cf22525dcb61443/download%0Ahttp://www.econ.upf.edu/~reynal/Civilwars_12December2010.pdf%0Ahttps://think-asia.org/handle/11540/8282%0Ahttps://www.jstor.org/stable/41857625