Performance Analysis of a Token Bucket Shaper for Different Communication Streams

By: Faheem, Fahd Humayun and Asif Mehmood

A thesis submitted to Department of Electrical and Electronics Engineering, University of Engineering and Technology, Peshawar for the Final Year Project on "Performance Analysis of Token Bucket Shaper for Different Communication Streams."

APPROVAL CERTIFICATE

This is to certify that the thesis submitted by Faheem, Fahd Humayun and Asif Mehmood is of sufficient standard to justify its acceptance by Department of Electrical and Electronics Engineering, University of Engineering and Technology Peshawar.

_____

Dr. Muhammad Inayatullah Khan Babar

Supervisor

Abbreviations

TB	Token Bucket
HTB	Hierarchal Token Bucket
DCF	Distributed Coordination Function
CSMA/CA	Channel Sensing Multiple Access/ Collision Avoidance
AP	Access Point
ISP	Internet Service Provider
QoS	Quality of Source
CBR	Constant Bit rate
VBR	Variable Bit rate
MPEG4	Motion Picture Experts Group Layer-4 Video
UDP	User Datagram Protocol
TCP	Transmission Control Protocol
IMS	IP Multimedia Subsystem
TNTC	Token bucket based Notification Traffic Control
MBTS	Measurement-Based traffic specification
VOIP	Voice over IP
NAM	Network Animator

ABSTRACT

The motive behind this thesis is to present the performance analysis of a Token Bucket Shaper for various communication streams. The study will provide a thorough insight into the Token Bucket Shaper and how its performance can be improved by adjusting its certain parameters. This study will provide an insight into how a node with Token Bucket Shaper will shape the real time traffic of MPEG4 video and real audio. This thesis will also show the effect of varying the parameters of the Token Bucket Shaper for improving its overall performance and making some trade-offs as per requirements.

The Token Bucket Shaper shapes the bursty and random data and makes it confined to the Network Agreement between the service provider and the user. The performance analysis of Token Bucket Shaper for Constant Bit Rates (CBR) and Variable Bit Rate (VBR) might have been done, however its performance analysis for MPEG4 video and real audio is lacking. Since MPEG4 video and real audio provides us with real world communication scenario such as video calls, video streaming etc.; a thorough insight into how the Token Bucket Shaper can assist in traffic shaping and making it sure that no packets are discarded due to the Network Agreement Violation is useful and mandatory. The performance analysis of the Token Bucket Shaper for various communication streams such as Audio, Video data will be of great importance in modern age computing world, since it assists in efficient service provision and Quality of Service (QoS) guarantee.

The methodology that was adopted in this project utilizes a single independent token bucket shaper for every node i.e. use of multiple independent Token Bucket Shapers for multiple traffic classes such as Audio and Video Data. We initially tested the token bucket shaper between two nodes where the source node generated MPEG4 data. Then in a separate scenario, we tested the Token Bucket Shaper for real audio data. Finally we used two independent Token Bucket Shapers for both real audio and MPEG4 data simultaneously generated from two different nodes, which was sent to a specific node.

The results of this analysis showed that the Token Bucket Shaper significantly shaped the bursty, and random data of the MPEG4 video. Though the delay was increased, but the packet loss was almost non-existing. The data was properly shaped and in agreement with the network parameters. The real audio data was not very bursty compared to the MPEG4 video, still Token Bucket Shaper efficiently shaped the real audio data and helped in getting rid of

the packet loss. Furthermore the variation in the parameters of the Token Bucket Shaper provided us with optimum performance of the Token Bucket Shaper.

The analysis presented encouraging results for providing the QoS guarantee. The Token Bucket Shaper's ability to shape the data and reduce the packet loss can help the network perform efficiently and reduce the violation of network agreement. Every user will be following the specified parameters and sending shaped data into the network which will save the network from malfunctioning.

Acknowledgements

We would like to express our deep and sincere gratitude to our honorable teacher and project supervisor, Professor Dr. Mohammad Inayat ullah Khan Babar whose guidance and encouragement enabled us to complete our project and thesis.

We would also like to thank our parents for their support and encouragement throughout our educational careers. We would also like to thank our other respected teachers for helping us throughout our Engineering Careers.

TABLE OF CONTENTS

TITLE PAGE

APPROVAL SHEET ……... i

LIST OF ABBREVIATIONS ……. ii

ABSTRACT …… iii

ACKNOWLEDGEMENTS …… v

TABLE OF CONTENTS ……..... vi

LIST OF FIGURES ……. ix

LIST OF CHARTS ……..... xi

Chapter 1: Introduction …… 1

1.1. Main Objectives ……..... 3
1.2. Justification ……..... 3
1.3. Limitations of the Project ……..... 4
1.3.1. Time Constraint ……..... 4
1.3.2. Expense of Hardware Implementation…….. 4

Chapter 2: Literature Review ……... 5

Chapter 3: Methodology …….. 13

3.1. Simulation Software ……13
3.2. Core Methodology …….. 14
3.3. MPEG4 Video …… 15
3.4. Real Audio Data ……. 17

Chapter 4: Simulations and Observations …… 19

4.1. Simulation Results for MPEG4 video …… 19
4.1.1. Simulation Results without Token Bucket Shaper …….. 19
4.1.1.1. Observations for Throughput …… 19
4.1.1.2. Observations for Delay ……. 21
4.1.1.3. Observations for Packet Loss ……... 21
4.1.2. Simulation Results with Token Bucket Shaper …….. 22
4.1.2.1. Observations for Throughput …….. 22
4.1.2.2. Observations for Delay ……... 23
4.1.2.3. Observations for Packet Loss ……. 24

4.1.3. Variation of the Token Bucket Shaper's Parameters for MPEG4 data. 25
4.1.3.1. Variation of Queue Length 25
4.1.3.1.1. Effect on Throughput 25
4.1.3.1.2. Effect on Delay 26
4.1.3.1.3. Effect on Packet Loss 26
4.1.3.2. Variation of Bucket Size 27
4.1.3.2.1. Effect on Throughput 27
4.1.3.2.2. Effect on Delay 27
4.1.3.2.3. Effect on Packet Loss 28
4.1.3.3. Variation of Token Generation Rate 29
4.1.3.3.1. Effect on Throughput 29
4.1.3.3.2. Effect on Delay 29
4.1.3.3.3. Effect on Packet Loss 30
4.2. Simulation Results for Real Audio Data 31
4.2.1. Simulation Results without Token Bucket Shaper 31
4.2.1.1. Observations for Throughput 31
4.2.1.2. Observations for Delay 32
4.2.1.3. Observations for Packet Loss 33
4.2.2. Simulation Results with Token Bucket Shaper 33
4.2.2.1. Observations for Throughput 34
4.2.2.2. Observations for Delay 35
4.2.2.3. Observations for Packet Loss 36
4.2.3. Variation of the Token Bucket Shaper's Parameters for Real Audio data 36
4.2.3.1. Variation of Queue Length.....37
4.2.3.1.1. Effect on Throughput.....37
4.2.3.1.2. Effect on Delay..... 37
4.2.3.1.3. Effect on Packet Loss 38
4.2.3.2. Variation of Bucket Size 39
4.2.3.2.1. Effect on Throughput 39
4.2.3.2.2. Effect on Delay 39
4.2.3.2.3. Effect on Packet Loss 39
4.2.3.3. Variation of Token Generation Rate 40
4.2.3.3.1. Effect on Throughput 41
4.2.3.3.2. Effect on Delay 41
4.2.3.3.3. Effect on Packet Loss 41
4.3. Simulation Results for both MPEG4 video and Real Audio data 43
4.3.1. Simulation Results without Token Bucket Shaper 43
4.3.1.1. Observations for Throughput 43
4.3.1.2. Observations for Delay 44
4.3.1.3. Observations for Packet Loss 45
4.3.2. Simulation Results with Token Bucket Shaper 46
4.3.2.1. Observations for Throughput 46
4.3.2.2. Observations for Delay 46

4.3.2.3. Observations for Packet Loss 48

4.3.3. Variation of the Token Bucket Shaper's Parameters for Real Audio and MPEG4 49

Chapter 5: Conclusion: Future Plans 50

Appendix A 52

Appendix B 57

Appendix C 63

LIST OF FIGURES

Figure-1: Leaky Bucket Algorithm 6
Figure-2: Token Bucket Algorithm 6
Figure-3: Multiple Correlated Token Bucket Shapers 7
Figure-4: Multiple Independent Token Bucket Shapers 15
Figure-5: The flowchart for shaping the MPEG4 video source 16
Figure-6: The flowchart for shaping the Real Audio Data 18
Figure-7a: The Network animator showing the random and bursty data 20
Figure-7b: The "bandwidth" utility showing the burstiness and randomness of MPEG4 data 20
Figure-7c: The throughput of the MPEG4 video without the use of Token Bucket Shaper 20
Figure-8: The delay vs time plot of unshaped MPEG4 video 21
Figure-9: Packet Loss for an MPEG4 video without using Token Bucket Shaper 21
Figure-10a: The NAM showing the shaped data 23
Figure-10b: The "bandwidth" utility showing the shaped MPEG4 data 23
Figure-10c: The throughput of the MPEG4 video with the use of Token Bucket Shaper ... 23
Figure-11: The delay vs time plot of shaped MPEG4 video 24
Figure-12: Packet loss for an MPEG4 video using Token Bucket Shaper 24
Figure-13a: The simulation scenario for real audio signal 31
Figure-13b: The throughput of unshaped real audio signal 31
Figure-13c: The throughput of the real audio without the use of Token Bucket Shaper 32
Figure-14: The delay vs time plot of unshaped real audio 32
Figure-15: Packet loss for a real audio data without using Token Bucket Shaper 33
Figure-16a: The simulation scenario for real audio signal with Token Bucket Shaper 34
Figure-16b: The throughput of shaped real audio signal 34
Figure-16c: The throughput of the real audio with the use of Token Bucket Shaper 35
Figure-17: The delay vs time plot of shaped real audio data 35
Figure-18: Packet loss for a real audio data using Token Bucket Shaper 36
Figure-19a: Simulation scenario for both MPEG4 and real audio data without Token Bucket Shaper 43
Figure-19b: The throughput of unshaped real audio and MPEG4 data 44
Figure-19c: The throughput of the real audio and MPEG4 video source without the use of Token Bucket Shaper 44
Figure-20: Delay vs time plot of unshaped real audio and MPEG4 data 45
Figure-21: Packet loss for real audio and MPEG4 video data without using Token Bucket Shaper 45
Figure-22a: Simulation scenario for both MPEG4 and real audio data without Token Bucket Shaper 47
Figure-22b: The throughput of shaped real audio and MPEG4 data 47
Figure-22c: The throughput of the real audio and MPEG4 video source with the use of Token Bucket Shaper 47

Figure-23: The delay vs time plot of shaped real audio and MPEG4 data …….. 48
Figure-24: Packet loss for real audio and MPEG4 video data using Token Bucket Shaper…… 48

LIST OF CHARTS

Chart-1: Throughput for varying Queue Length 25
Chart-2: Delay for varying Queue Length 26
Chart-3: Packet loss for varying Queue Length 26
Chart-4: Throughput for various Bucket Sizes 27
Chart-5: Delay for various Bucket Sizes 28
Chart-6: Packet loss for various Bucket Sizes 28
Chart-7: The throughput for different Token Generation Rate 29
Chart-8: Delay for various Token Generation Rates 30
Chart-9: Packet loss for varying Token Generation Rate 30
Chart-10: The effect of varying the Queue Length on the throughput 37
Chart-11: The effect of varying the Queue Length on delay 38
Chart-12: The effect of varying the Queue Length on packet loss 38
Chart-13: Variation of the throughput with the Bucket Size 39
Chart-14: Variation in delay with varying Bucket Size 40
Chart-15: Variation in packet loss with varying Bucket Size 40
Chart-16: Effect of varying Token Generation Rate on throughput 41
Chart-17: Effect of varying Token Generation Rate on delay 42
Chart-18: Effect of varying Token Generation Rate on packet loss 42

Chapter 1

Introduction

The humans have a natural tendency to communicate with each other. This communication with each other establishes a human network. The computer was invented in order to perform tasks quicker and more efficiently than the human beings. It alone was able to do certain tasks, however to make it work even better; the idea of networking came into the world of computers as well. The basic motive behind networking is to allow the quick and reliable sharing of information between different computers and other electronic devices through any channel forming a network. Internet is one of the gifts of the networking of computers as well.

The users of computers can communicate with other users sitting thousands of kilometers away, surprisingly at a faster speed. Online games, live streaming, and advanced audio/video communications are also the due to the advancement of computer networking. The introduction of demanding applications need better networking technology, which can support faster data rates, avoid packet losses, and provide efficient results. Researchers aim at providing better services at lower cost and making less trade-offs.

Last couple of decades witnessed a great rise in the world of computers. Computers are no longer limited to large corporations. It has become the part of every household. Other electronic equipment such as cellular phones, PDA's, Tablets, Ipad's, IPods and other portable devices have taken over the whole market. The introduction of wireless technologies has revolutionized the whole market. It does provide a lot of comfort and facilities, but it puts ever increasing burden on the network which is used by all these electronic devices. The increase in the number of users also increases the load on the network to main the Quality of Service and entertain every user as per there need. The competition in the world of computing is now based on provision of the best services at lowest prices. The race is no longer only of what services can you provide, it is also about the quality of services as well. Every user wishes to have the best possible audio, video etc. services. However that is dependent upon the cost as well. For better services, one has to pay higher charges. Imagine a user paying less but sending more than allowed data into network, which totally violates the bilateral agreement between the user and the network. How would the network know if there is

violation of the agreement? What even if the network knows that the user is violating the agreement and sending "non-conformant" packets? The packets will be discarded which will be loss of the user. What should the user do to make sure that the network agreement is not violated so that no data should be discarded? There is a need for some mechanism which has to take care of above mentioned issues.

The user and the network are in a bilateral agreement which must be taken care of at all cost. The agreement involves various parameters which both the user and the network have to follow. This is in the mutual advantage of both the user and the network. For users, the packets which are in violation of the network agreement will be discarded if the network has any policing algorithm. That's why the user has to make sure that the data which it sends is not violating the agreement otherwise it will suffer from packet loss. What it means is that there is a need for traffic regulation on both ends. This regulation is either traffic policing or traffic shaping.

The policing algorithm at the network end is usually either Leaky Bucket Algorithm [1]. At the user end, in order to shape the data; we need the Token Bucket Shaper which works on Token Bucket Algorithm.

The Token Bucket Algorithm consists of a fixed size bucket in which tokens, generated at some rate, are stored. The tokens are continuously generated and stored in the bucket. A packet which is to be sent to the network will only be sent if there are enough tokens available in the bucket. In case there is a deficiency of tokens, then the packet has to wait in a queue. Tokens are consumed when the packets are sent into the network. For a packet which has "n" bytes, "n" will be consumed as well for sending it into the network [2]. The bucket size has to be greater than the maximum packet size otherwise the packet will be in the queue and will never be sent.

The Token Bucket Shaper deals with controlling the traffic. The advantage of the Token Bucket Shaper is that it helps in reducing the packet loss, and minimizing the jitter as well. The Token Bucket Shaper based on the Token Bucket Algorithm provides traffic shaping for Quality of Service Guarantee. It shapes the traffic as per network characteriastics and help the user get rid of data loss. The sent data follows the network parameters. However this might cause some delay since the packet might have to wait for tokens. The token bucket shaper has certain parameters such as bucket size, queue length and the token generation rate. All these factors influence the performance of the token bucket shaper in one way or the

other. There is an optimum value of these parameters available for a specific network. Simulations can be used to find about the optimum value and then use it for the specific token bucket shaper. The token bucket shaper operates at the IP layer; however it can work between the MAC layer and IP layer [5] which reduces the need to modify the existing IEEE 802.11 MAC.

Up till now, most of the performance analysis done related to the Token Bucket Shaper is based on the use of Constant Bit rate (CBR) sources and there is a lack of a thorough information related to the performance analysis of the Token Bucket Shaper for real world scenarios which is usually bursty and random data. The CBR fails to replicate the real world random and bursty data such as audio and video. There is a need for presenting analysis based on real world data sources.

1.1 Main Objective:

As mentioned earlier that there is lack of the information related to the performance analysis of token bucket shaper for real world data sources.

The main objectives are:

- To analyze the performance of the token bucket shaper for different communication streams such as real world audio and video data
- To find the optimum performing parameters of the token bucket shaper.

1.2 Justification:

The way a token bucket shaper will perform against these real world data in a simulation test will indicate how good the token bucket shaper can shape the data. The simulation model will present us with the necessary tools for playing with different parameters of the token bucket shaper in order to be able to achieve the optimum performance of the Token Bucket Shaper. There is need for coming up with the real world scenario for analyzing the Token Bucket Shaper.

1.3. Limitation of the Project:

1.3a Time Constraint: Time is a major issue in a project like this. Since there is a lot of room for improvement in the token bucket shaper model as well as a lot of work can be done related to the analysis of the token bucket shaper. So time is a major constraint.

1.3b Expense of Hardware Implementation: Hardware implementation is an expensive approach for the token bucket shaper. If finances are available then the analysis can be performed in an even better way.

Chapter 2 presents a review of the literature and work carried on related to Token Bucket algorithm and Token Bucket Shaper. Chapter 3 will present the methodology adopted in this project and thesis. Chapter 4 will present the simulation results as well as present the performance analysis of the token bucket shaper. Chapter 5 will conclude the thesis and highlight the future plans.

CHAPTER 2

Literature Review

Chapter 2 will present the review of the work carried on Token Bucket Algorithm and token bucket shaper. Numerous research papers have been written on the use of Token Bucket algorithm, its significance and how it can be utilized for the benefit of the user. There has been some work done related to the token bucket shaper as well. Researchers are interested in improving the efficiency of the token bucket algorithm and token bucket shaper. Major issue is related to the parameters such as queue size, the rate at which the tokens are generated, the bucket size etc. Material is also present regarding how to determine the parameters of the token bucket efficiently without any significant cost.

Nguyen Hong Van in his Doctoral Thesis on "Mobile Agent Approach to Congestion Controlling Heterogeneous Networks" writes that the traffic shaper can control the data and its flow into the network. The basic advantage of the traffic shaper is that it reduces the packet loss, causes a decrease in the latency and queue delay and lowers the jitter for the system. The location where the shaper has to be installed is the edge of network. There are two algorithm for traffic shaping namely Leaky Bucket and Token Bucket algorithm. Leaky bucket allows bursty data when there are sufficient tokens available. In token bucket algorithm, if the tokens arrive when the bucket is full; then the tokens will be discarded. The packet will be sent into the network only when there are sufficient tokens available. If a packet arrives and sufficient tokens are not available for the packet then such packet is known as non-conformant. The token bucket shaper will treat such packet as one of the three basic methods

- The packet will be dropped.
- It is tagged as "non-conformant" and will be dropped if the network becomes overloaded.
- The packet waits in a queue till sufficient tokens are available in the bucket.

Figure-1 presents a sketch of leaky bucket algorithm. The leaky bucket is based on the idea of a bucket which has hole at its bottom. There is a constant flow of water (data) into the bucket which seeps down through the hole. In case the bucket is full and further data comes into it then that data will be discarded. The hole at the bottom of the bucket guarantees constant flow. The problem with the leaky bucket algorithm is that it lacks the bit rate saving

capability. However it does not allow any burst of data. In short, the leaky bucket algorithm is based on the leakage of packet at a specific leaky rate.

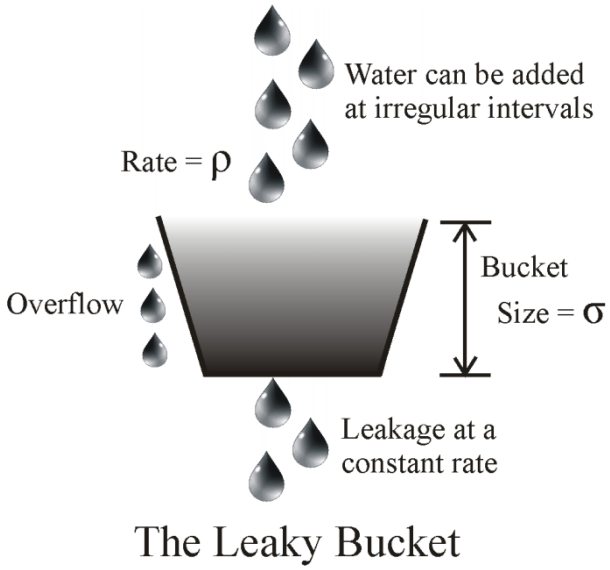

Figure-1: Leaky Bucket Algorithm

Figure-2 shows that the token bucket algorithm is based upon the use of a token generator which generates token at a rate "ρ". The generated tokens are stored in the token bucket of size "σ". A packet is conformant only if there are sufficient tokens available in the bucket for the packet.

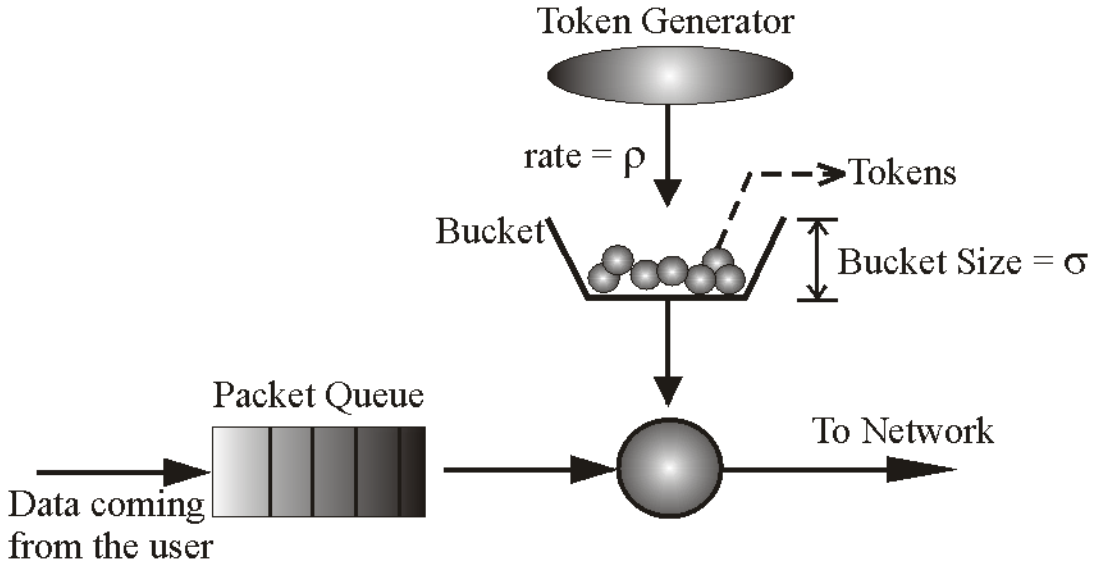

Figure-2: Token Bucket Algorithm

T.H Lee in his paper titled “Correlated Token Bucket Shapers for Multiple Traffic Classes” states that data sent from different nodes can be shaped by using either Multiple Correlated Token Bucket Shapers or using multiple independent token bucket shapers. The use of multiple correlated token bucket shapers provides us with the flexibility of using the unused bandwidth of other traffic class however multiple correlated token bucket shapers are complex to implement. The independent token bucket shapers lack the flexibility of multiple correlated token bucket shapers but its implementation is easier compared to its counterpart.

Figure-3 presents a sketch of how the multiple correlated token bucket shapers can work. It can be clearly seen that there are multiple token generators. These generated tokens are placed in a token placement module which places the token in the respective bucket as explained in [1].

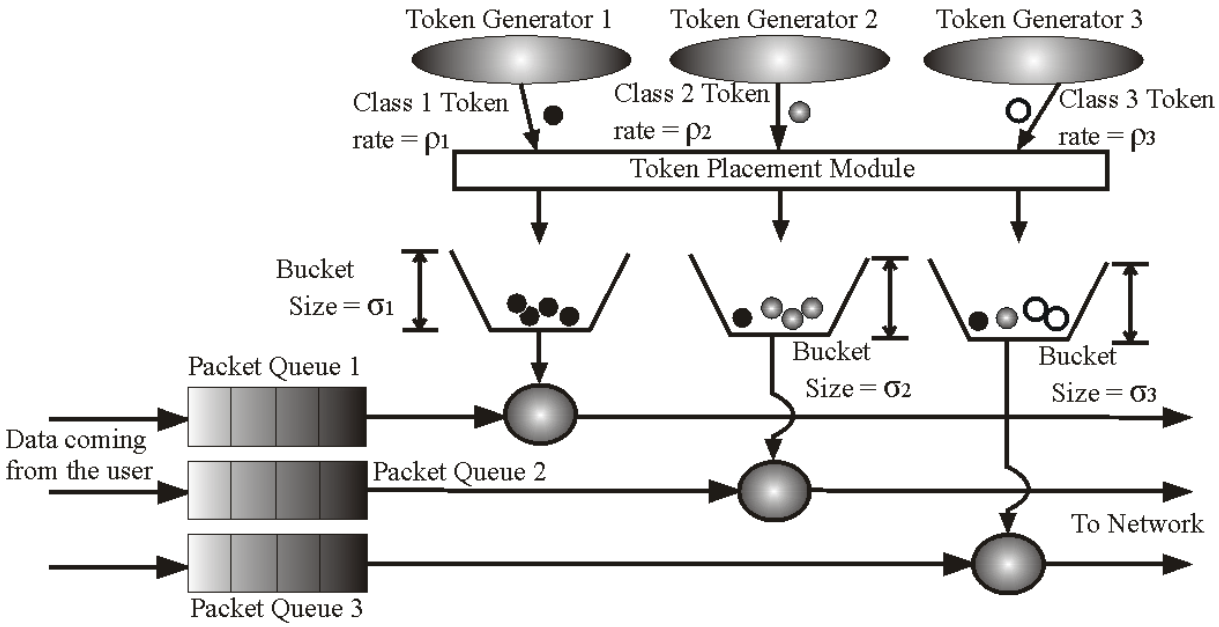

Figure-3: Multiple Correlated Token Bucket Shapers

Iman Shames et al. [3] presents the idea of training an intelligent agent for learning about the token generation of the Token Bucket at various network states. This will result in efficient utilization of bandwidth as well preventing the overloading of the traffic in various other parts of the network. This will cause a decrease in the number of packets dropped in the whole network. The simulation results are quite satisfactory as packet dropping probability is

low and the packet injection into the network is increased by decreasing the used buffer size in every router for keeping the delay (caused by large buffer/ queue size) as small as possible. They have divided the routers into Network Routers and Source Routers. The source routers are those which are connected to the end nodes while the network routers are those which are not directly connected to the end nodes. Every source routers has a token bucket which has a specific bucket size and token generation rate.

Wilfried N. Gansterer et al. [4] have used the token bucket for preventing the Spam E-mails at the outgoing SMTP server. The basic motive behind such idea is to reduce the outgoing unwanted emails that are sent through any network, since too many emails which are usually sent by any spammer can cause an extra load on the network. The token bucket is used as a central component in dynamically controlling the outgoing emails. The proper use of this technique will significantly reduce the spam emails, while it will not affect the regular e-mails. The mentioned method can eliminate the spam email messages hence reducing the network load as well as providing the customers with convenience. The mentioned techniques can be used to deal with the following two scenarios.

- The Token Bucket component limits the flow of outgoing email from a network "X". The proper care of the parameters will cause no effect on the regular outgoing emails.
- A whitelist or blacklist component will not allow the outgoing emails to go through any other channel except the server "X"

The significance of the work of Wilfried N. Ganstere et al. is that token bucket was usually concerned with traffic shaping, however their work has brought out certain points which are attractive in terms of avoiding spam emails. Its implementation and handling is convenient. The consumption of the tokens for dealing with the spam email messages is based upon the size of the spam email. Their approach has certain advantages over other used methods as it is more flexible and adaptive in limiting the outgoing emails. The most important property of this approach is that it targets only the spammers and there is no effect on regular mail users.

The token bucket shaper has inherent bit rate saving capability which is not found in the leaky bucket algorithm.

David Perez et al. [5] present the performance analysis of the Token Bucket Shaper on an IEEE 802.11 real test bed. The testing of such equipment is cumbersome as there is a need

for some standard. If a token bucket shaper is inserted between the MAC layer and the IP layer, then there is no need to modify the existing 802.11 MAC. Their parameters are controlled on the basis of the information which is obtained through the lower layers in the stack. The traffic handling and bandwidth management mechanisms lay the foundations for the QoS mechanism. They present the experimental performance analysis of the Token Bucket Shaper on real test bed of about 11 mobile stations. The use of the Token Bucket Shaper reduces the standard deviation of the throughput and smoothens the competition among the stations for the channels; however the trade-off is in the form of reduction of the mean global throughput. Adaptation of the TBF parameters to the application will facilitate us i.e. we can trade-off some throughput for limited delay and less jitter. Depending upon the priorities, one can optimize the throughput and simultaneously reduce the standard deviation. Limiting the station which damages the system performance can reduce the effect of the performance anomaly of the IEEE 802.11. The benefits are not very satisfactory with TCP compared to the UDP as TCP has its own congestion control algorithm. The proposed solution can also be applied to IEEE 802.11e, but fi the traffic flows belong to the same category then the 802.11e will not behave accordingly. For such cases, one can use the Hierarchical Token Bucket.

Partha Kanuparthy et al. [6] talks about measurement methods for end-to-end detection of traffic shaping. The Shaperprobe is used as an end-to-end detection mechanism which can detect the presence of a traffic shaping in a particular path. The Shaperprobe can also estimate the shaper's characteriastics if there is any. The results obtained from the deployment of shaperprobe on large scale on M-Lab is also presented which indicates the presence of traffic shaping in certain Internet Service Providers. This approach is the first design detection methodology which can detect and measure the traffic shaping deployment in the Internet. The results indicate that Internet Service Providers are leaning towards the deployment shaping and policing mechanism in order to manage various distributed applications which demand high resources. The results also indicated that the usually small set of shaping configurations is deployed by the ISP's. There were also certain ISP's which did not advertise about shaping in their services; however they did shape the traffic

J.L. Valenzuela et al. [7] highlight the use of a Hierarchical Token Bucket Shaper in a combination with IEEE 802.11 WLAN's. The analysis provides an insight into the advantages of the proposed scheduler. The simulation shows that when multiple hosts, all of

which transmitting at various rates; the throughput of the higher rated hosts falls below the level of the lower rate hosts. This indicates two issues which are

- The average throughput of all the hosts is imposed by the lowest bit rate host.
- The standard deviation attains high values

The cause for the first issue is the DCF in CSMA/CA as it allows equal channel probability for every host. In case when a host with lower rate controls the channel, then the station punishes the hosts which transmit for longer time than the faster hosts do. Hence there is a need for a solution which can deal with traffic from different hosts independently in order to provide a sustained throughput to each and every host.

The solution for above mentioned issues is the use of a HTB for constraining the traffic in hosts in correlation with the Access Point for getting the required QoS effects. The simulation shows that TCP has less available bandwidth because of the ACK frames, while UDP has more bandwidth available. The HBT is available at the IP layer and it controls the manner in which the packets are forwarded to the MAC layer. This approach does not depend upon the wireless standard used. The QoS can be provided in WLAN's which constrains the channels to a lower bit rate. This approach results in a sustained throughput which has low standard deviation to stations and even to differentiated services.

Jianxin Liao et al. [8] present the idea of using the token bucket based notification traffic control (TNTC) mechanism for dealing with the heaving signaling load on IP Multimedia Subsystem (IMS) network caused by notification traffic. It is an application layer solution deployed at presence server. The motive behind the TNTC is upgrading the valid access probability in the meanwhile controlling the notification message traffic. The main probability features are analyzed and the effect of various parameters on the performance of TNTC is also investigated. The simulation results showed that the notification traffic can be effectively controlled by the TNTC. The performance also got better in comparison to the existing schemes in terms of valid access probability and update arrival rate.

First the design of proposed approach is discussed. After that the queuing theory is used to model the TNTC and calculate its main probability feature. Analytical theory is used to study the effects of various parameters on the TNTC performance, and on the basis of that; presented the general guidelines for these parameter settings. The TNTC with fixed and Exp

methods are also compared via simulations which indicate that TNTC works better than other methods in terms of valid access probability and the updated arrival rate.

Puqi Perry Tang et al [9] highlight different problems which might arise in the derivation of the parameters of the token bucket. The parameters can be obtained by using the computer network's flow of traffic patterns in two scenarios. The scenarios are

- The token bucket parameter set is identified for the flow of data such that all the data are delivered immediately without introducing any delay or packet loss.
- A queue is added with the token bucket model so that the "non-conformant" packets are stored until there are sufficient tokens available in the bucket.

The addition of queue makes the traffic smoother which makes the resulting parameters of the token bucket less demanding and its fluctuations decrease as well with the passage of time. The relationship between the size of the queue and parameters of token bucket is thoroughly analyzed. The delay caused due to queuing of every packet is also presented which can be used to adjust the traffic pattern after the queuing. Algorithm for deriving the Measurement-Based traffic specification (MBTS) is also highlighted which gets rid of the laborious job of characterizing the traffic in advance for reservation.

The problem with the currently used Token bucket model for traffic shaping is that the token bucket parameters have to be found out from the traffic pattern. In case when delay is not desired and the target is to directly send the data into the network, bucket size (b) and token generation rate (r) has to be found out. The addition of the queue will have the above mentioned effects such as smoothening the data flow.

The optimal parameters of the token bucket shaper are obtained by observing the flow of the data. The use of MBTS technology will expectedly make the use of RSVP easier, which will speed up the deployment of internet QoS services. The inherent adaptability and precision of the MBTS improves the network efficiency by assisting the network applications in subscribing the appropriate amount of network resources.

Tsern-Huei Lee [10] et al. highlights the use of rate monotonic algorithm with traffic shaper for the provision of QoS guarantee in ATM networks. Work has been done related to the admission control in case of CBR, however for bursty traffic; there is need for an efficient criterion as the CBR criterion can reduce the utilized bandwidth. The utilized approach is based upon the use of a token bucket regulator for regulating or shaping the VBR source.

R. Bruno et al. [11] talks about the procedure for evaluating the Linear Bounded Arrival Processes (LBAP) traffic characterization parameters when the traffic flow follows the stochastic model. The basis for this procedure is the use of a queuing system applied to a VoIP scenario. The network architecture is of Differentiated Services.

Mohammad F. Alam et al [12] highlights that the source's traffic parameters must be defined in form of token bucket traffic descriptors before the guaranteed service flow can be set up. The responsibility of the source is to act as per the requirement of the descriptors by traffic shaping. The thorough study of the token bucket shaper's effect on the characteristics of transmission of MPEG video over the guaranteed service is presented. The manner in which the parameters of the shaper affect the transmission performance for various types of MPEG video is also presented.

Chapter 3

Methodology

Chapter 3 presents the methodology adopted for this thesis. In section 3.1, the used simulation software is discussed along with its features and specialties. Section 3.2 highlights the core methodology which is based on using multiple independent token bucket shapers for shaping the data generated by every node. Section 3.3 presents the discussion on the methodology adopted to get the MPEG4 video data as an input to the nodes. Section 3.4 highlights the methodology used for inputting the real audio traffic into our simulation software.

For the performance analysis of the token bucket shaper, we used the NS-2 simulation software. Some information related to the NS-2 is presented in below.

3.1. Simulation Software:

The Network simulator 2 or NS-2 is open source simulation software that works as event-driven simulator. It was designed for research related to computer communication networks. Throughout the years NS-2 has gained interest from various fields of life ranging from industries to academia and government. It is continuously growing and including many new protocols. There are many modules now integrated into NS-2 like transport layer protocols, routing protocols, and application etc.

It is an object oriented simulator which is written in C++ and it used the OTcl as the front end interpreter. NS-2 follows class hierarchy both in C++ and OTcl. From user's point of view, there is a close tie between the two hierarchies [13].

The reason behind using two languages is that the simulator has to do two different types of things. Since there is a need for an efficient system programming language which can manipulate the packets, headers etc. for the simulation of various protocols. Such operations require a faster run time speed, while the importance of the turn-around time is less.

On the contrary, there is need for varying parameters or some configuration for a huge portion of the network research. In such scenarios, the iteration time has much more

significance. The configuration runs mostly once so there is less importance of the run time as well.

The network simulator satisfy both the requirements by using two languages C++ and OTcl as C++ can be run faster however it is slowly changed which makes it ideal for the implementation of detailed protocol. On the other hand, OTcl has a slower run time but it can be altered very quickly which makes it the perfect choice for simulation configurations.

Now the question arises which language should be used for what? The answer to this question is simple. Use OTcl for the setup, configuration and things that will be used only once, while use the C++ language in cases where there is need for processing every packet that is the part of the network flow. However there can be exceptions such as when your task involves invoking the OTcl code many times, then we better switch our code to C++.

There is still work in progress about including new modules into the NS-2 and there is a lot of potential for its growth.

3.2. Core Methodology:

The methodology used in this thesis is based upon the use of individual, independent token bucket shaper for every data source. Every token bucket shaper has its own bucket, its own token generator which is totally independent of the other token bucket shaper. We could have also adopted the method of using multiple correlated token bucket shapers due to its flexibility; however it is much more complex to implement it [1]. The parameters of the individual token bucket shaper are in the user control which can be adjusted as per needs. The significance is that the user will be able to make certain trade-offs as per his/her need by tweaking the values of token generation rate, token bucket size, the queue size etc. All these parameters will provide the user with desired results.

The basic motive is to shape the data which will reduce the packet loss, smoothen the throughput and improve the delay. The parameters of the token bucket shaper has to be adjusted as per the data source i.e. there is no hard and fast rule to have specific parameters. It is based upon the user need and the trade-offs that one is willing to make. For every data source, there is a specific value of the each token bucket parameter which will give the optimum performance.

Figure-4 shows the core methodology adopted. The two independent token bucket shapers each with its own queue, token bucket and token generator is shaping the traffic of its own respective source. One of the token bucket shaper is shaping the MPEG4 video while the other one is shaping the traffic of the real audio traffic generator.

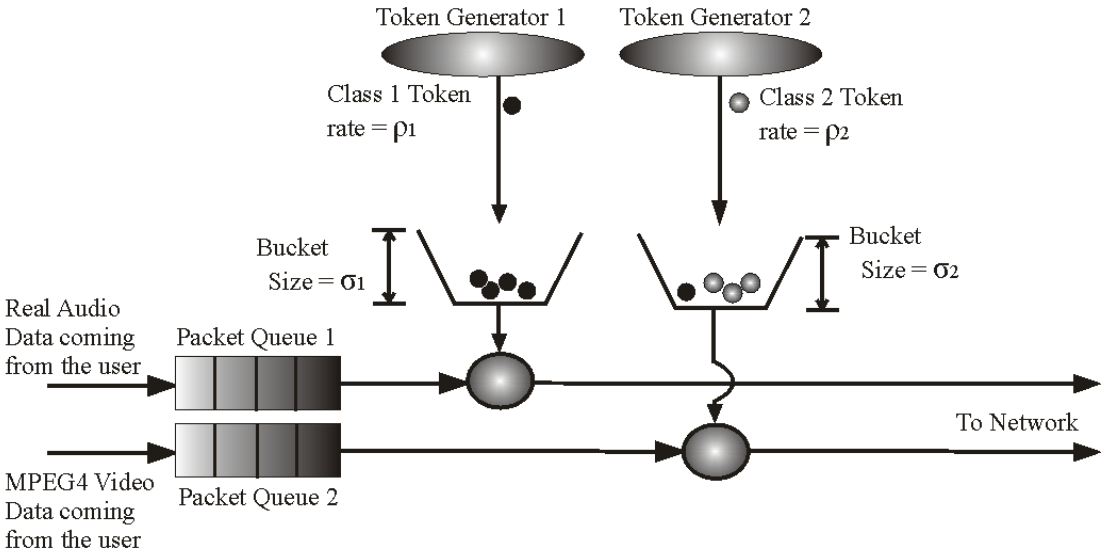

Figure-4: Multiple Independent Token Bucket Shapers

In order to perform the analysis of the token bucket shaper for real time data, we used the static data of MPEG4 video and real audio which is presented below.

3.3. MPEG4 Video:

For the MPEG4 video analysis, this thesis uses the Video Traffic Generator which relies on a specific model of MPEG4 trace files known as the Transform Expand Sample (TES) Model. The contributors of these video traffic generators are Ashraf Mathrawy, and Ioannis Lambadaris. This traffic generator produces a traffic whose first and second order statistics are just like the original MPEG4 trace file. Hence this is the video source which is different the VBR and CBR sources available in the NS-2. This video traffic generator replicates the original MPEG4 video; hence the performance analysis of this video traffic generator is the same as that of any other MPEG4 video sources.

The video traffic generator is attached to a node along with a UDP agent. Initially there is no token bucket shaper attached to the node for shaping the data. So the unshaped

data goes from one node to another without any regulation. There can be packet loss expected along with random and bursty data. In order to get rid of the packet loss and smoothen the data, the token bucket shaper is attached to the node which generates the video traffic. The token bucket shaper is an independent shaper whose parameters can be adjusted accordingly.

The part of code that utilizes the MPEG4 is

set n1 [$ns node] #declare a node
set udp0 [new Agent/UDP] # declare a new node
$ns attach-agent $n1 $udp0 # attach the agent to the node
set video [new Application/Traffic/MPEG4] # video traffic generator
$video set initialSeed_ 0.4
$video set rateFactor_ 5
$video attach-agent $udp0 # attach udp0 agent to the video source
$ns at 1.8 "$video start" # traffic generator starts working

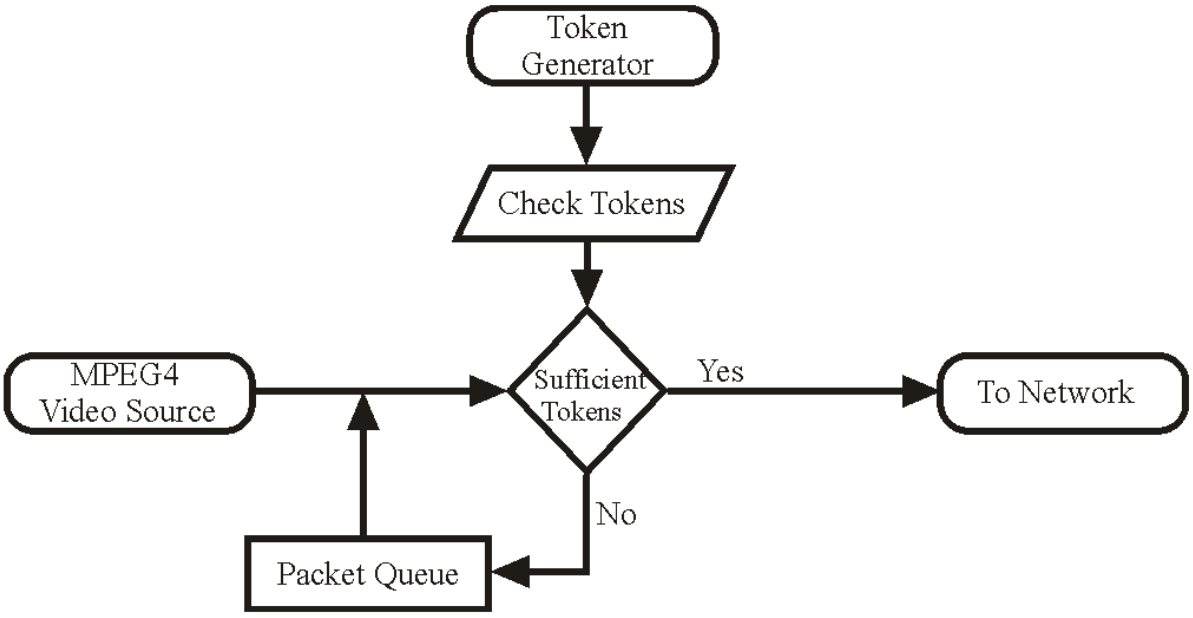

Figure-5: The flowchart for shaping the MPEG4 video source

Figure-5 summarizes the whole process. The MPEG4 video data before being sent to the network has to be checked whether the packets are conforming or not. For that the tokens are checked, if there are sufficient tokens present in the bucket; the packets will be forwarded

to the network which indicates that the packets are conforming. If the tokens are not sufficient it means that the packets are not conforming and the packets have to wait in the queue till sufficient tokens are available.

3.4. Real Audio Data:

The second phase of the analysis is based upon analyzing the real audio traffic. This thesis uses the real audio trace files which are available at broadcast.com. The C++ code is provided by University of Southern California[1*]. The CBR and VBR sources in NS-2 cannot be used as the alternative sources for real world audio data. The real audio traffic replicates the characteristics of real world audio signals i.e. it contains the real audio signals in static data form.

Finally the MPEG4 video and real audio data was sent to a network through two different sources. First their throughput, delay and packet loss was noted without the use of Token Bucket Shaper. Then a token bucket shaper was inserted and the corresponding values of throughput, delay and packet loss were noted.

The audio traffic generator is attached to a node along with a UDP agent. Initially there is no token bucket shaper attached to the node for shaping the data. So the unshaped data goes from one node to another without any regulation. There can be packet loss expected along with random and bursty data. In order to get rid of the packet loss and smoothen the data, the token bucket shaper is attached to the node which generates the audio traffic. Since the token bucket shaper is an independent shaper, we can adjust its parameters accordingly.

Figure-6 presents the flow chart for the real audio traffic data. The real audio data before being sent to the network has to be checked whether the packets are conforming or not. For that the tokens are checked, if there are sufficient tokens present in the bucket; the packets will be forwarded to the network which indicates that the packets are conforming. If the tokens are not sufficient it means that the packets are not conforming and the packets have to wait in the queue till sufficient tokens are available.

[1] The code is available at http://www-rp.lip6.fr/ns-doc/ns226-doc/html/realaudio_8cc-source.htm as well at the appendix of the thesis

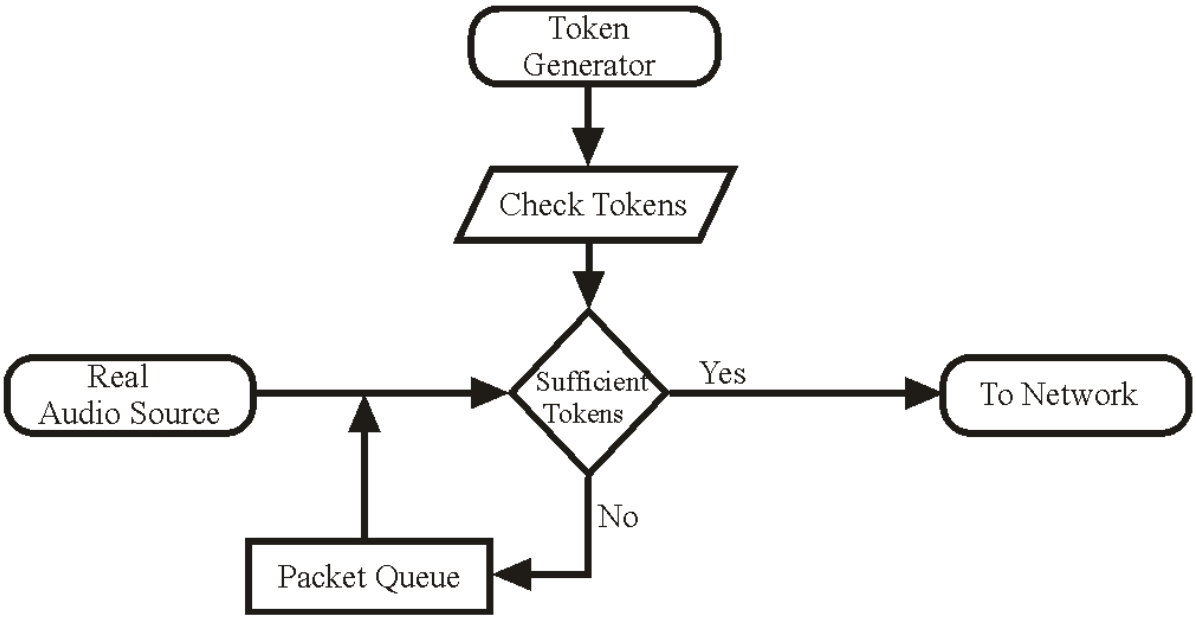

Figure-6: Flowchart of shaping the Real Audio Data

The part of code which utilizes real audio is

set realaudio [new Application/Traffic/RealAudio]

$realaudio set packetSize_ 100

$realaudio set burst_time_ 0.05ms

$realaudio set idle_time_ 1800ms

set udp0 [new Agent/UDP]

$udp0 set fid_ 1

$udp0 set rate_ 32.768k

$udp0 set bucket_ 1024

$realaudio attach-agent $udp0

Chapter 4

Simulations and Observations

Chapter 4 presents the simulations and the observations collected as a result of the simulation in NS-2. There are three different scenarios in which the simulation was carried out and the results were noted. For every scenario, the token bucket parameters were varied and the corresponding readings were noted down. The observations mostly collected were related to the throughput, delay and the lost packets.

Section 4.1 will present the 1st scenario of the simulation which is analyzing the performance of the token bucket shaper for MPEG4 video along with the observations. Section 4.2 will highlight the 2nd scenario of the simulation which is based upon the analysis of the token bucket shaper for a real audio signal along with the observations. In section 4.3, the 3rd scenario of simulation in which both the MPEG4 video data and real audio data coming from different nodes is shaped by multiple independent token bucket shapers is presented along with the observations.

4.1. Simulation Results for MPEG4 video:

The simulation environment consists of two nodes in which node '0' acts as the source node while node '1' acts as the sink. The link between the nodes was 2.20 Mbps with 100ms delay. The UDP agent is connected to node '0' along with the MPEG4 video source.

4.1.1. Simulation Results without Token Bucket Shaper: The simulation was first run for 20 seconds without any token bucket shaper so the NAM showed the two nodes in which the data was unshaped while the bandwidth window between the two nodes showed the unshaped traffic. The Xgraph showed the delay, bandwidth or throughput and packet loss.

4.1.1.1 Observations for throughput: The throughput showed that the data was random and bursty. There was sufficient delay and packet loss. Figure-7a shows the simulation scenario where node '0' is transmitting the MPEG4 data to node '1' without any token bucket shaper installed at node '0'. Figure-7b shows the random and bursty data by using the "bandwidth" function available in the Nam. Figure-7c shows the throughput of MPEG4 video without the use of a token bucket shaper using Xgraph utility of NS-2. It can be clearly seen that the data is random and bursty.

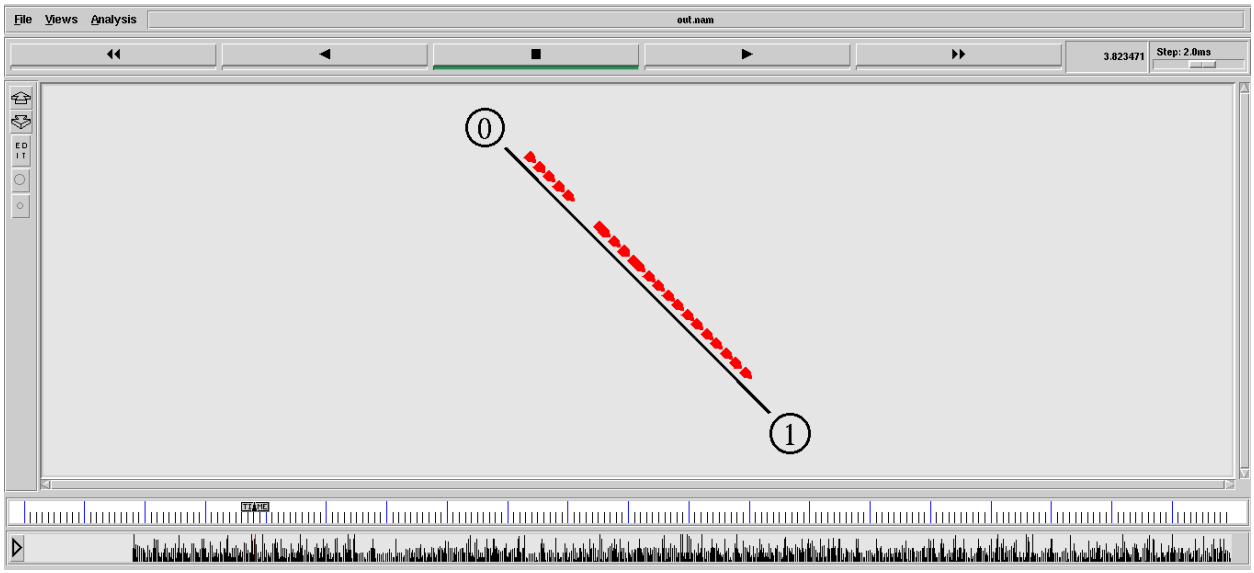

Figure-7a: The Network animator showing the random and bursty data.

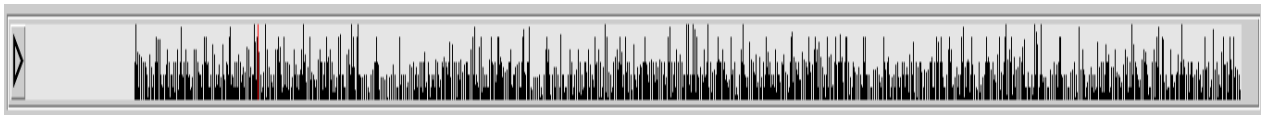

Figure-7b: The "bandwidth" utility showing the burstiness and randomness of MPEG4 data.

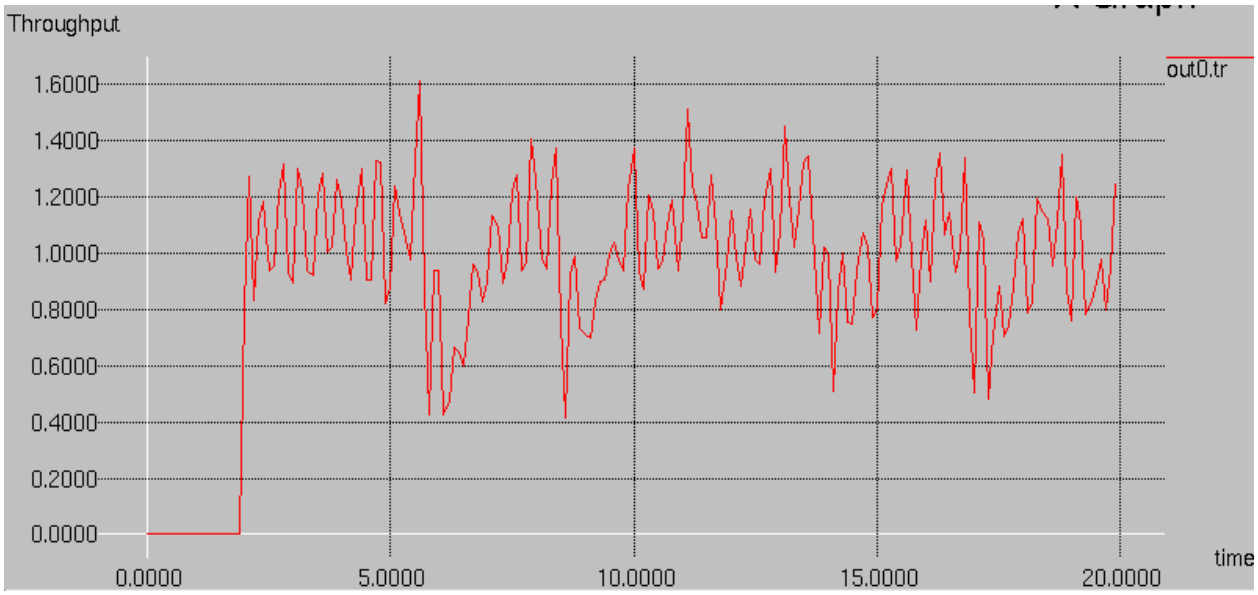

Figure-7c: The throughput of the MPEG4 video without the use of token bucket shaper.

4.1.1.2 Observations for delay: Using the above mentioned simulation scenario, the delay that was observed is shown in figure-8. It can be seen that initially there is a huge delay which reduces significantly later.

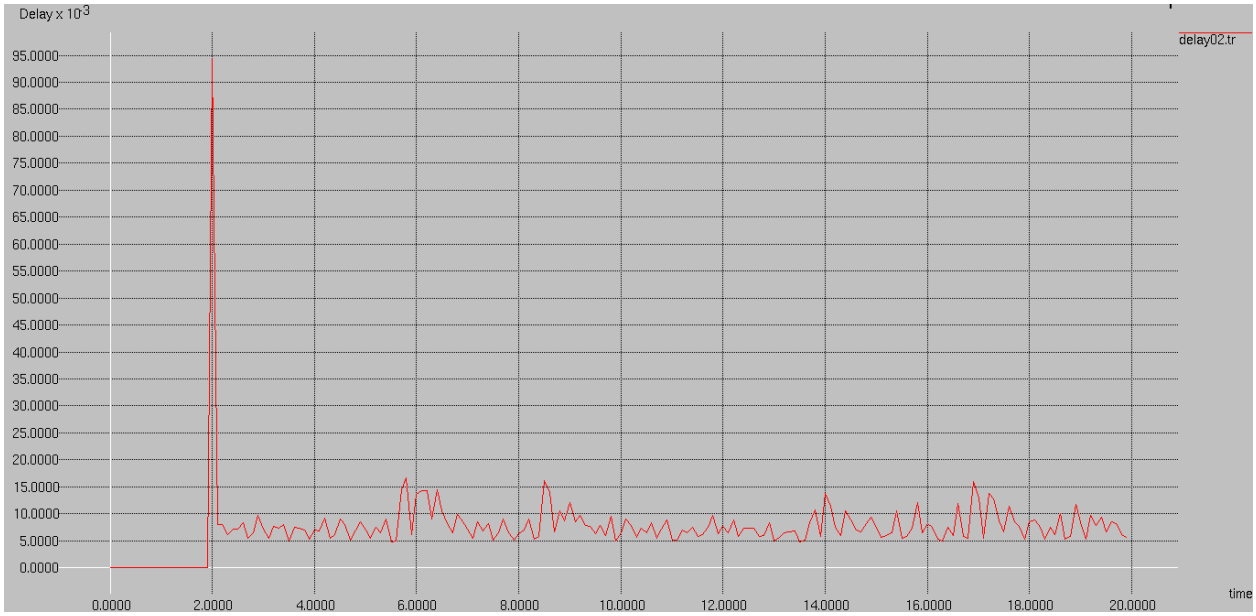

Figure -8: The delay Vs time plot of unshaped MPEG4 video

4.1.1.3 Observations for Packet Loss: There is a significant packet loss due to the random and bursty data. Since these packets are non-conformant to the network parameters, they are discarded. The loss of packets can be troublesome specially in voice and video communications.

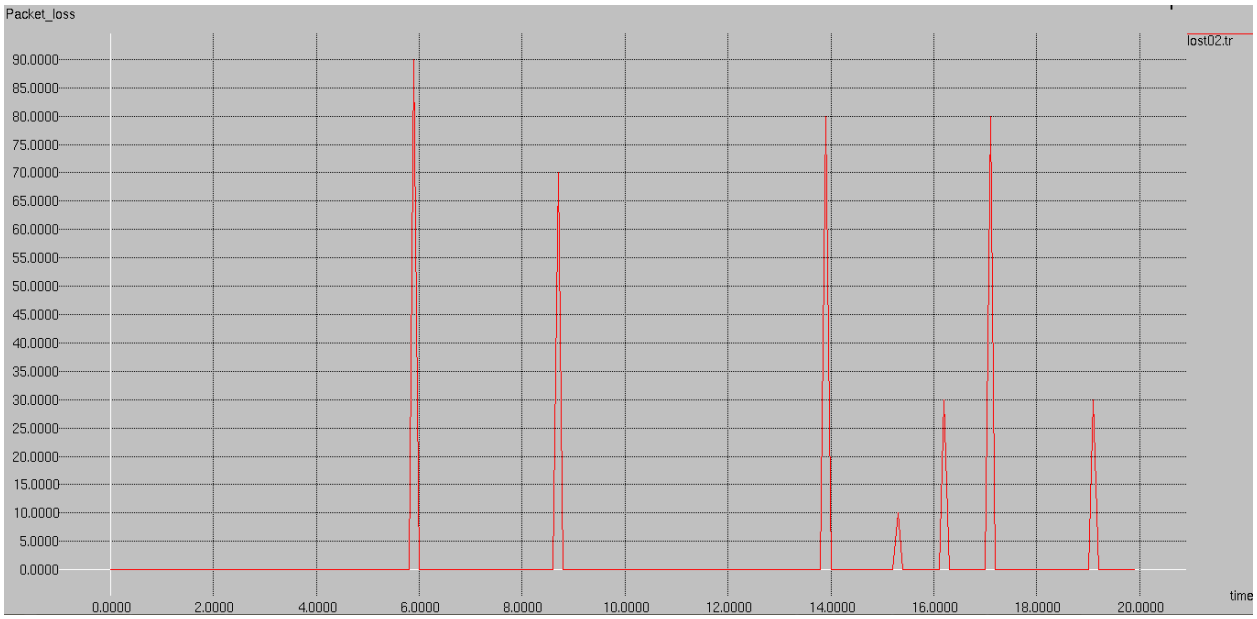

Figure-9: Packet loss for an MPEG4 video without using token bucket shaper.

Figure-9 shows the packet loss that was observed for the MPEG4 video without the use of token bucket shaper. There are significant numbers of packets dropped due to the burstiness of the data.

4.1.2. Simulation Results with Token Bucket Shaper: The simulation was first run for 20 seconds with the token bucket shaper attached to the node. The NAM showed the two nodes in which the data was shaped while the bandwidth window between the two nodes showed the shaped traffic. The Xgraph showed the delay, bandwidth or throughput and packet loss.

After attaching the token bucket shaper to the node '0', the MPEG4 video was smoothened. The bursty and randomness was taken care of. The parameters used are as follows

- Bucket Size is 1024 Bytes.
- The token generation rate is 32.768k
- The queue length is 1000

The data is shaped as per network agreement, which tackles the issue of packet loss and the QoS guarantee is provided.

The effect of token bucket shaper on throughput, delay and packet loss is discussed in the subsections below.

4.1.2.1. Observations for Throughput: The use of token bucket shaper smoothens the data sent. The MPEG4 video which was bursty and random as seen in Figure-7a, Figure-7b and Figure-7c becomes smoother by using the token bucket shaper. Figure-10a shows the Nam output of the shaped data. In contrast with Figure-7a, the data is shaped as per the network agreement and there are no bursts and randomness. Figure-10b shows the Nam bandwidth bar, which is totally different from figure-7b. The burstiness in figure-7b vanishes in figure-10b due to the use of the token bucket shaper. Figure 10-c shows the X-graph throughput of the shaped MPEG4 video. The output in this case is better than the one seen in figure-7c. Hence the performance in terms of throughput gets better by using the token bucket shaper.

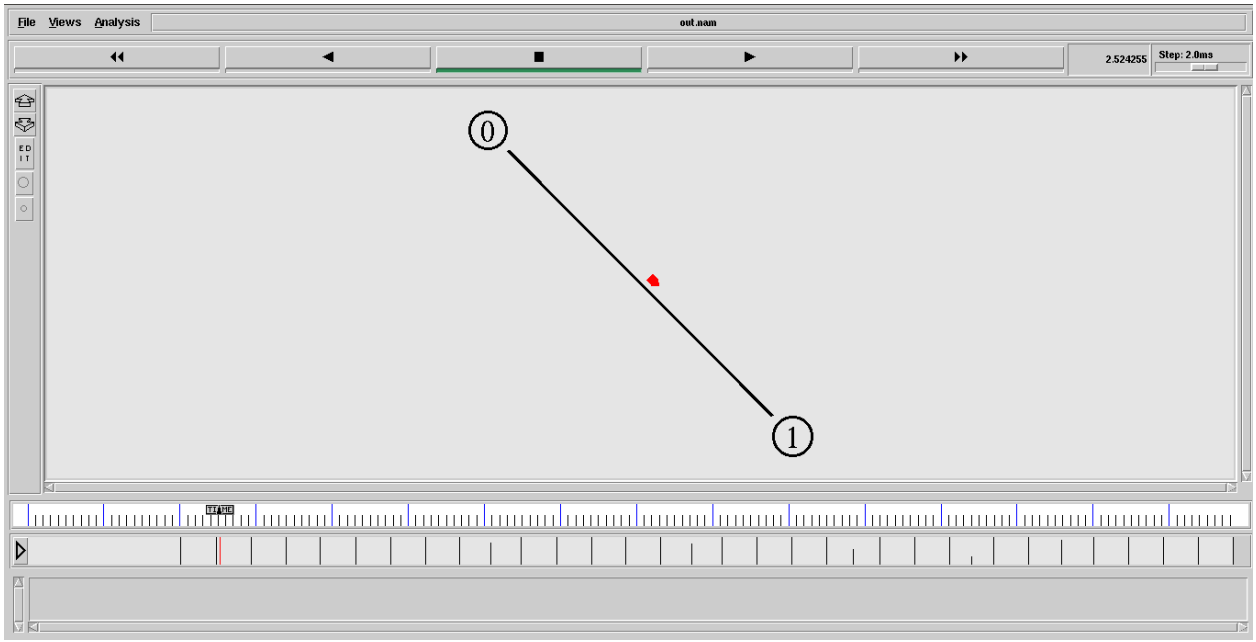

Figure-10a: The NAM showing the shaped data

Figure-10b: The "bandwidth" utility showing the shaped MPEG4 data

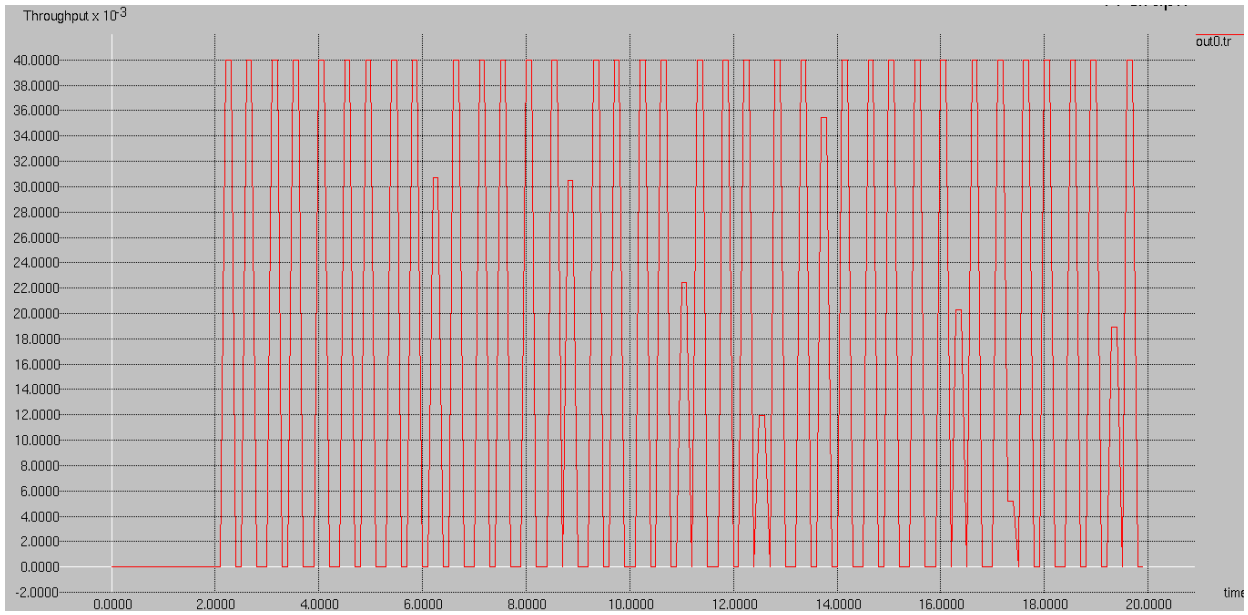

Figure-10c: The throughput of the MPEG4 video with the use of token bucket shaper.

4.1.2.2 Observations for delay: The delay increased by using the token bucket shaper. The reason behind that is the data has to wait in queue till sufficient tokens arrive. However the user can willingly make trade-off for better system performance i.e. increases the delay but smoothen the data.

Figure-11 shows the increase in delay due to the use of the token bucket shaper. The delay has significantly increased compared to figure-8.

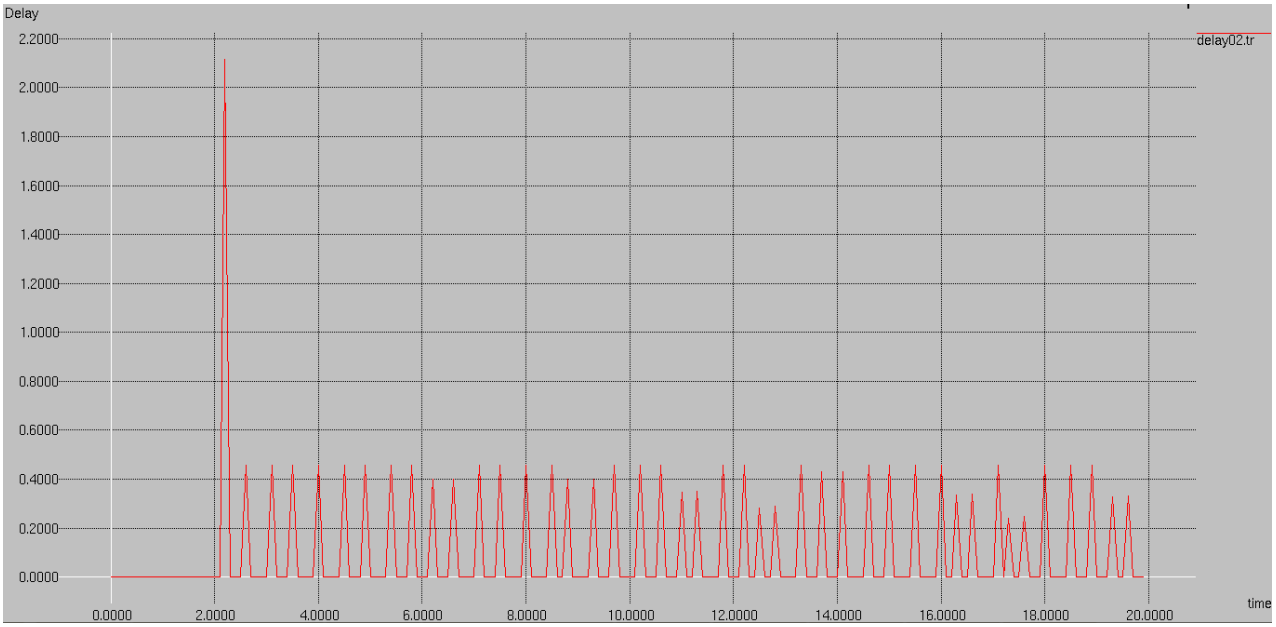

Figure-11: The delay Vs. time plot of shaped MPEG4 video

4.1.2.3. Observations for Packet Loss: The most significant fact observed about the performance of the token bucket shaper is that the packet loss is reduced. In the simulations carried out for this thesis, it was observed that there is no packet loss with the use of the token bucket shaper.

Figure-12 shows the Xgraph of the packet loss obtained as a result of simulation. The result was quite satisfactory as the packet loss was a major issue earlier in the case when there was no token bucket shaper used as depicted in figure-9.

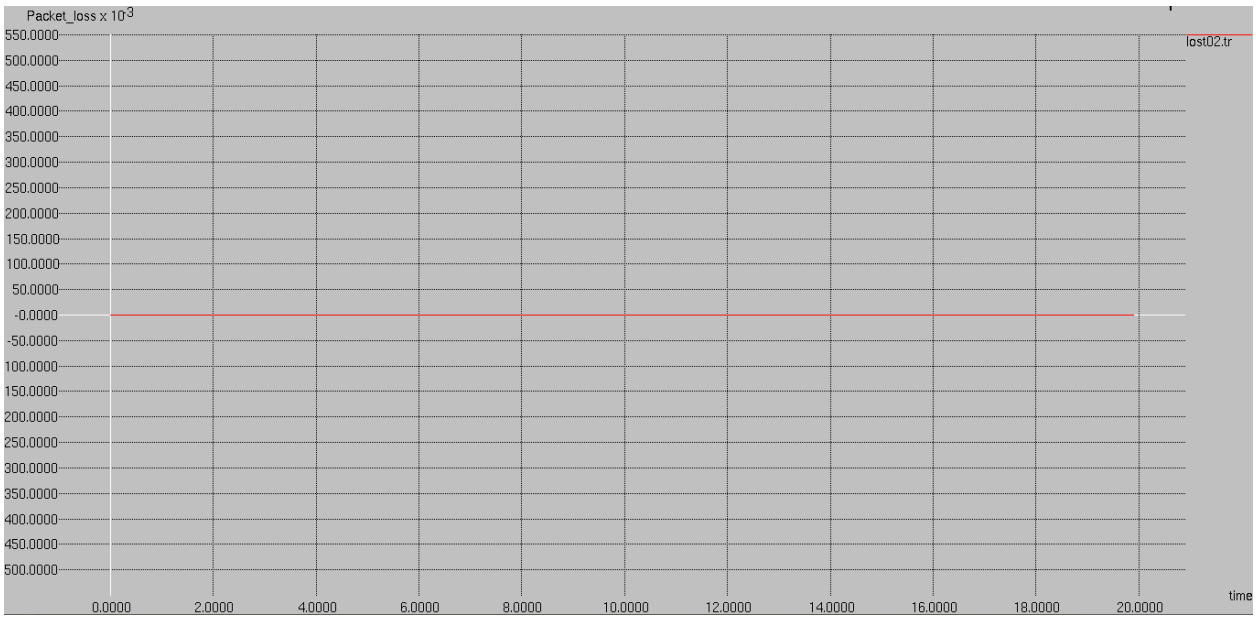

Figure-12: Packet loss for an MPEG4 video using token bucket shaper.

4.1.3 Variation of the Token Bucket Shaper's Parameters for MPEG4 data: This section presents the effects of varying different parameters of token bucket shaper on its performance. The parameters are bucket size, queue length and token generation rate. The thorough analysis of the effects of varying these parameters will be presented.

4.1.3.1. Variation of Queue Length: In the above mentioned simulation scenario we used three different values of queue length which were 10, 100, 1000. Variation of queue length has different effects on the throughput, delay and packet loss which are presented in the subsections.

4.1.3.1.1. Effect on Throughput: The burstiness in throughput increases with queue length=10 after 8.7 seconds, while it was same for queue length of 100 and 1000 which means that the queue length for optimum performing token bucket shaper for this MPEG4 video data in terms of throughput is 100. Making it 1000 will not affect the throughput however it will cause added hardware as the queue length has to be increased.

Chart-1 shows the effect of varying queue length on throughput which is in agreement with our statement.

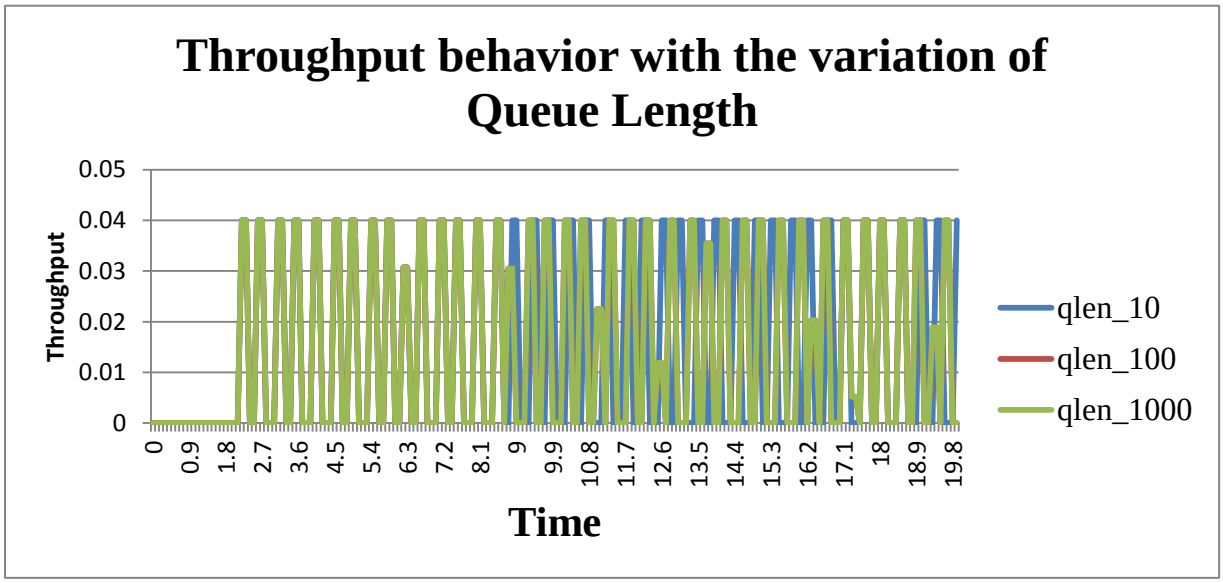

Chart-1: Throughput for varying Queue Length

4.1.3.1.2. Effect on Delay: Variation in the value of queue length did not affect the delay at all, which indicated that in order to improve the delay performance of the token bucket shaper; there is no need to alter the value of the queue length.

Chart-2 backs the above statement as there is no effect on Delay by changing the Queue Length.

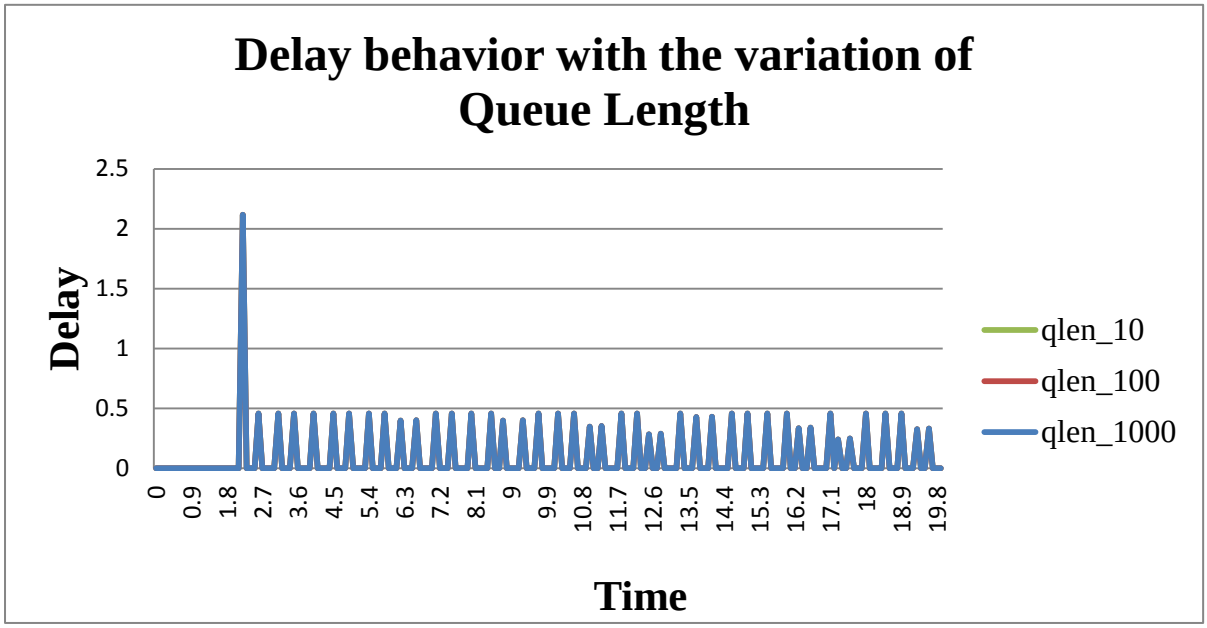

Chart-2: Delay for Varying Queue Length

4.1.3.1.3. Effect on Packet loss: The packet loss for queue length of 100 and 1000 is non-existent however for queue length of 10, a huge packet loss occurs after 6.5 seconds, which indicates that the optimum performing queue length is 100 in terms of packet loss as well.

Chart-3 provides the evidence based on simulation for the above statement.

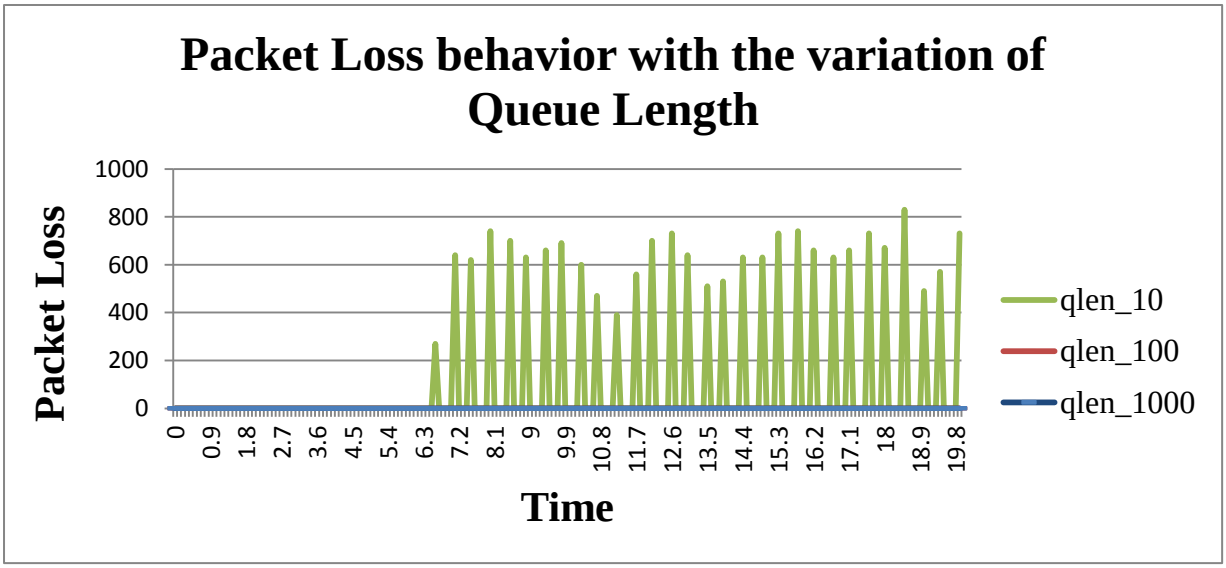

Chart-3: Packet loss for varying Queue Length

4.1.3.2. Variation of Bucket Size: In the above mentioned simulation scenario we used three different values of bucket size which were 512, 1024, 2048. Variation of bucket size has different effects on the throughput, delay and packet loss which are presented in the subsections.

4.1.3.2.1. Effect on Throughput: The effect of bucket size on throughput is a bit amazing.

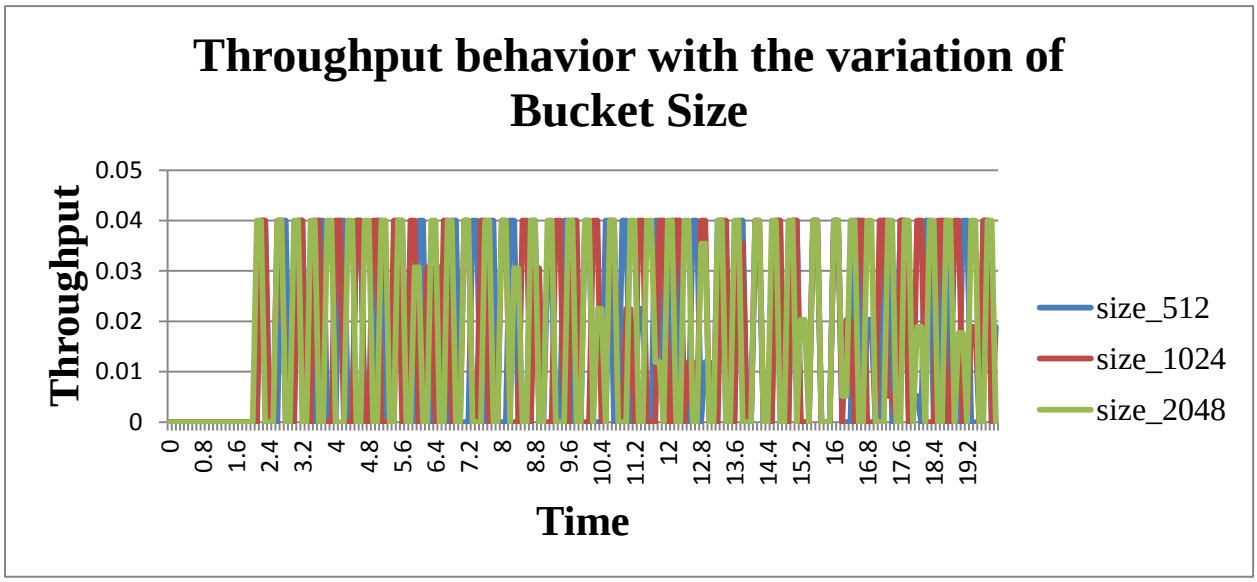

Chart-4: Throughtput for various Bucket Size

There is no significant effect on throughput except that increasing the bucket size makes the packet reach the sink i.e. node '0' in this case, faster indicating the improved delay performance.

Chart-4 proves the above statement.

4.1.3.2.2. Effect on Delay: The effect of bucket size on delay happens on small scale, i.e increasing the bucket size slightly improves the delay performance. Simulation showed that if improved delay is desired then increasing the bucket size is one of the choices we can make.

Chart-5 shows that increasing the bucket size improves the delay performance slightly.

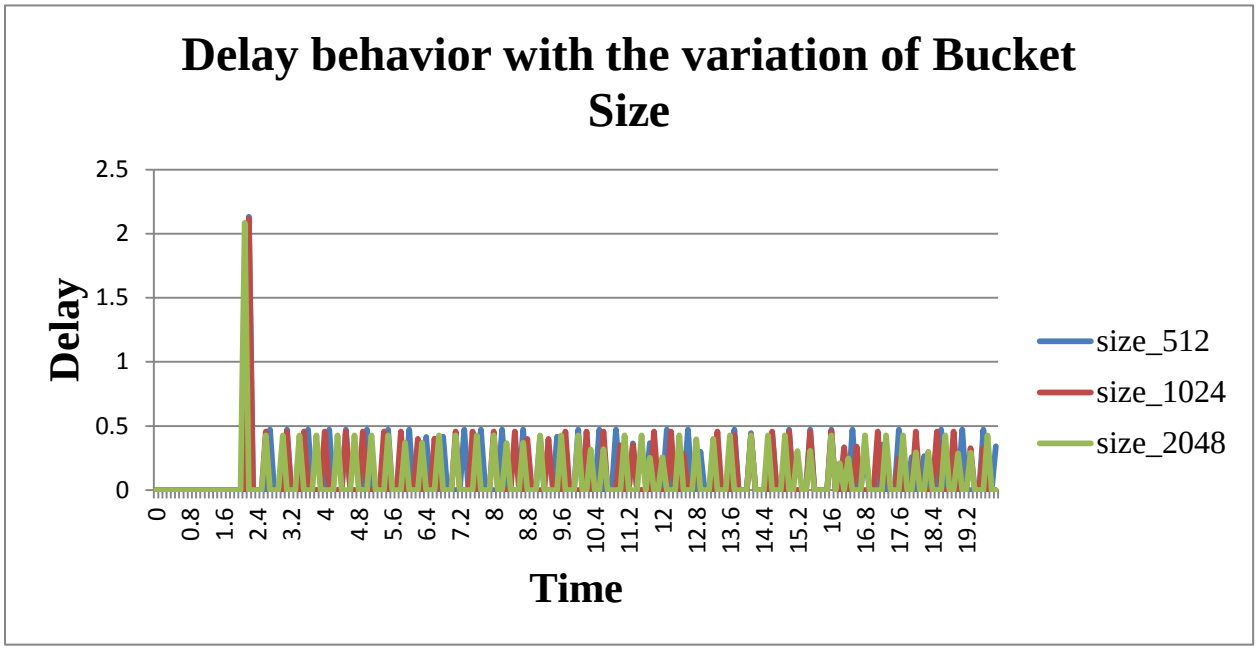

Chart-5: Delay Behavior for various Bucket Size

4.1.3.2.3. Effect on Packet Loss: Variation of the bucket size does not affect the packet loss at all. The packet loss remains as it was i.e. in this case the packet loss is zero due to the token bucket shaper.

Chart-6 provides the graph showing the packet loss for various token bucket size.

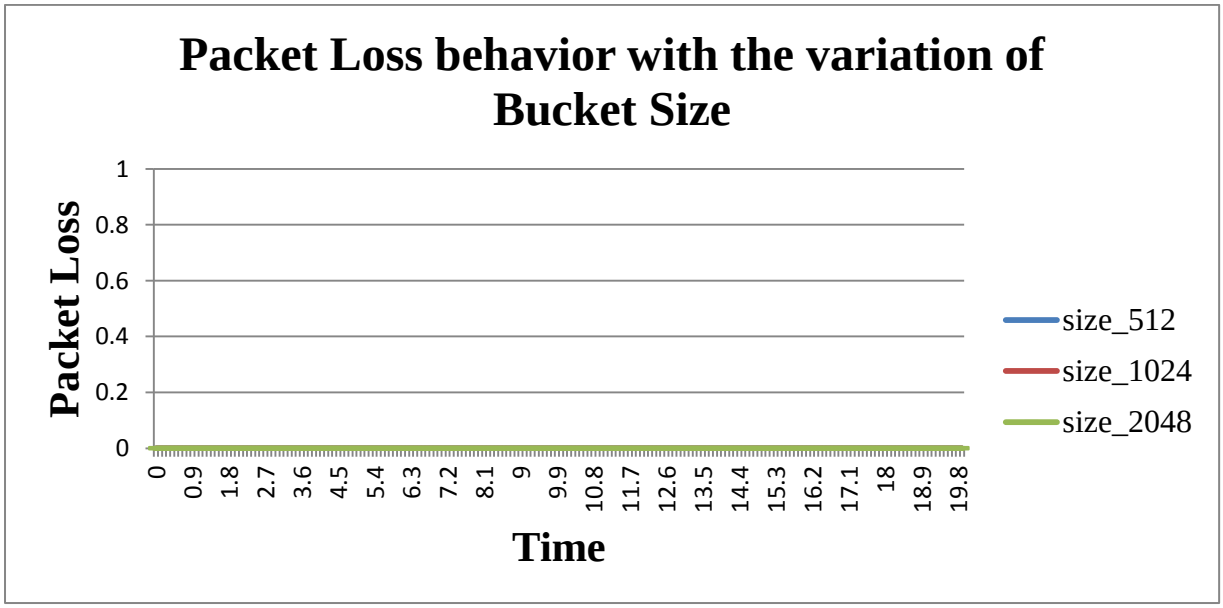

Chart-6: Packet loss for various Bucket sizes

Base on the above results, the bucket size for the optimum performing token bucket shaper for the MPEG4 video data is 2048.

4.1.3.3. Variation of Token Generation Rate: In the above mentioned simulation scenario we used three different values of token generation rate which were 16.384k, 32.768k, and 65.536k. Variation of token generation rate has different effects on the throughput, delay and packet loss which are presented in the subsections.

4.1.3.3.1. Effect on Throughput: Increasing the token generation rate will increase the burstiness and randomness. The burstiness might be favored only if it doesn't cause any packet loss. However increasing the token generation rate too much will jeopardize the use of token bucket shaper as the data will be extremely bursty and will cause loss of packets.

It can be clearly seen from Chart-7 that the curve with the lowest token generation rate has the least burstiness and randomness.

4.1.3.3.2. Effect on Delay: Increasing the token generation rate will cause a decrease in the delay. This is clearly understood as a higher token generation rate means less waiting for the packet to be transmitted which will cause less delay.

Chart-8 shows that the token generation rate of 65.536k will cause the least delay.

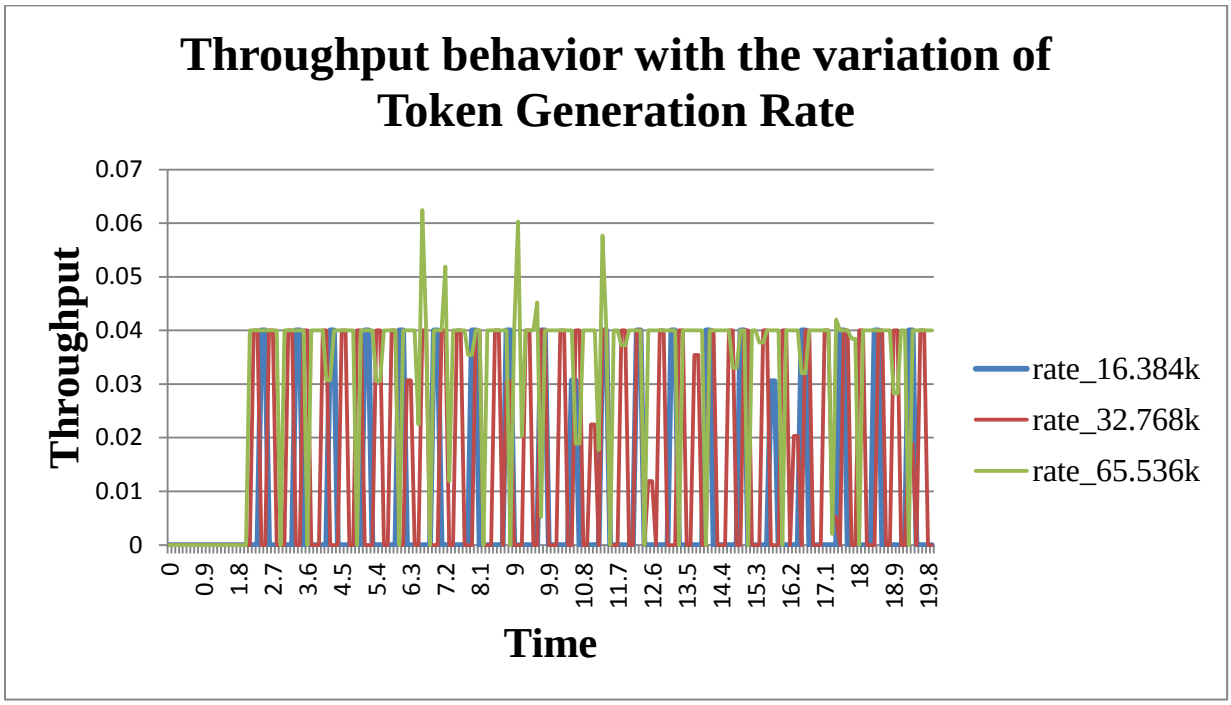

Chart-7: The throughput for different token rate generation.

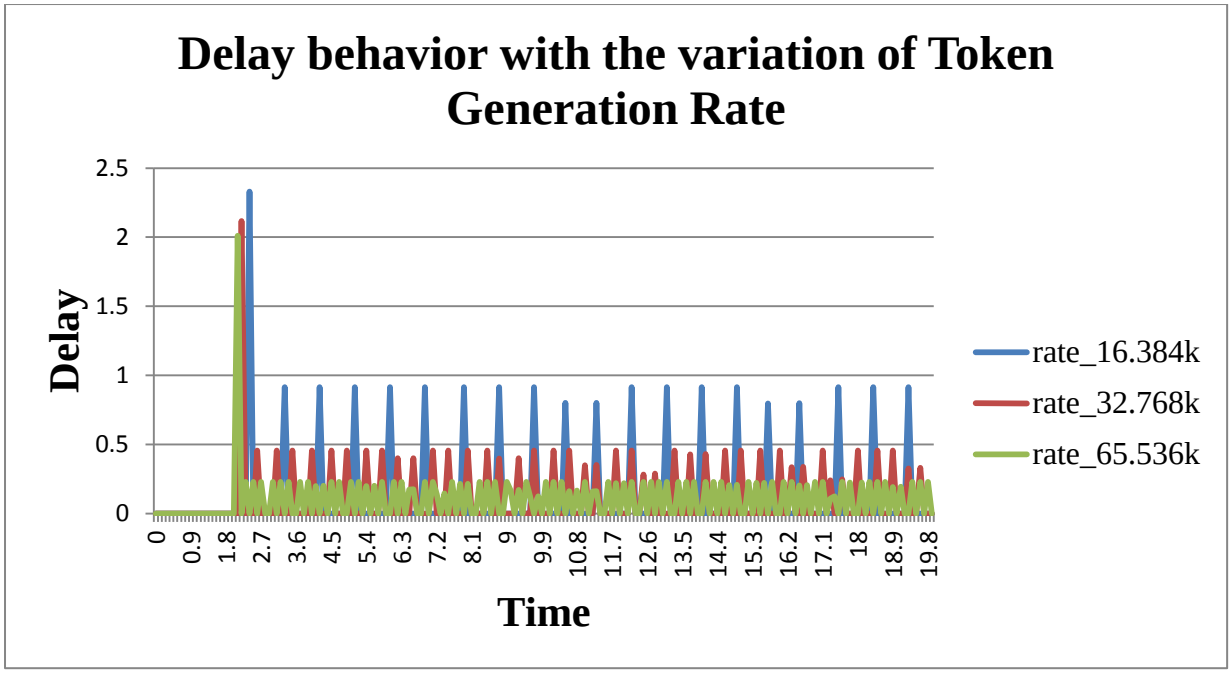

Chart-8: Delay for various Token generation rate

4.1.3.3.3. Effect on Packet loss: Increasing/decreasing the token generation rate in this case did not cause any change in the packet loss, however it is clear that if token generation rate increases too much then the data will become bursty and there will be packet loss.

Chart-9 shows the packet loss for varying values of token generation rate.

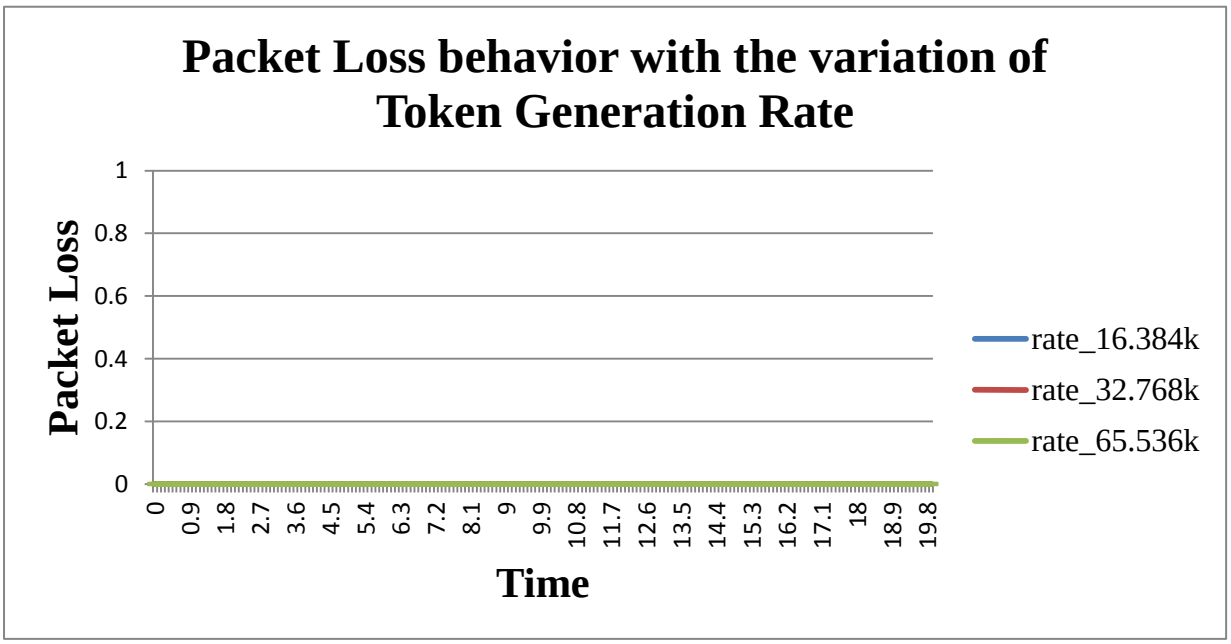

Chart-9: Packet loss for varying Token Generation Rate.

4.2. Simulation Results for Real Audio Data:

The simulation environment consists of two nodes in which node ‘0’ acts as the source node while node ‘1’ acts as the sink. The link between the nodes was 2.20 Mbps with 100ms delay. The UDP agent is connected to node ‘0’ along with the real audio data source.

4.2.1. Simulation Results without Token Bucket Shaper: The simulation was first run for 20 seconds without any token bucket shaper so the NAM showed the two nodes in which the data was unshaped while the throughput window between the two nodes showed the unshaped traffic. The Xgraph showed the delay, bandwidth or throughput and packet loss.

4.2.1.1 Observations for throughput: In order to show the throughput of the real audio signal which is unshaped, this thesis utilizes the NAM, the bandwidth option in NAM and the Xgraph utility available in the NS-2. Figure-13a presents the simulation scenario in NAM for an unshaped real audio signal. Figure-13b shows the NAM bandwidth bar where the straight lines represent the real audio signal. It can be seen that even the unshaped/un-regulated real audio signal pronounces lesser randomness and burstiness compared to the MPEG4 video.

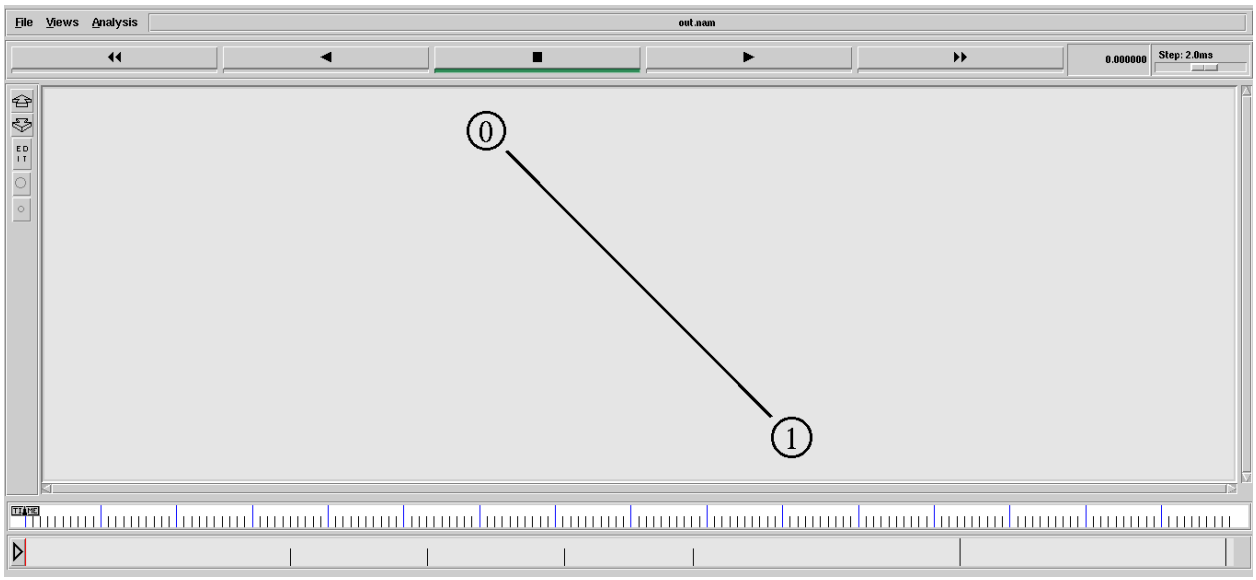

Figure-13a shows the simulation scenario for real audio signal

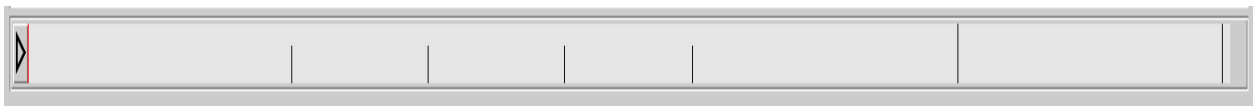

Figure-13b shows the throughput of unshaped real audio signal

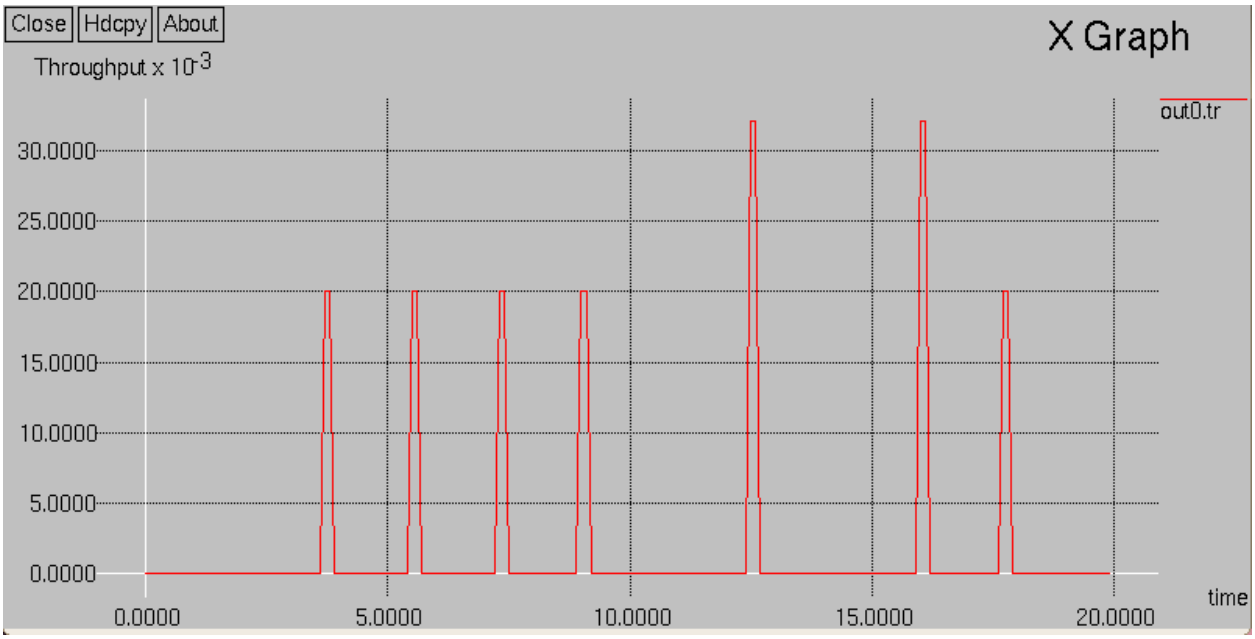

Figure-13c: The throughput of the real audio without the use of token bucket shaper.

Figure-13c shows the throughput using the Xgraph utility. The data is not smooth and there are some bursts but they are less compared to the MPEG4 video.

4.2.1.2 Observations for delay: There is some delay present even without the use of token bucket shaper. Figure-14 presents the sketch of delay which is present during transmission of the real audio signal from node '0' to node '1'.

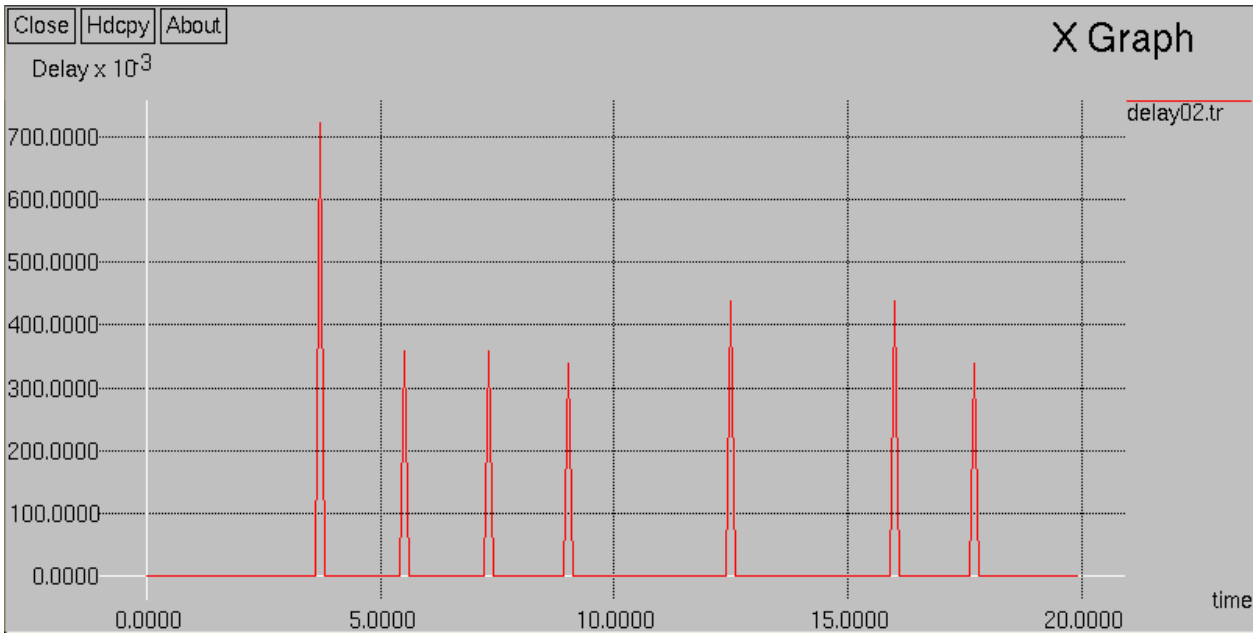

Figure-14: The delay vs time plot of unshaped real audio

4.2.1.3 Observations for Packet Loss: Since the real audio data used for this thesis is not that bursty and random so it does not violate the network agreement and we observe no packet loss even in the absence of the token bucket shaper which indicates that all the packets are conforming. Figure-15 presents the packet loss that occurs during the transmission of unshaped real audio signal with respect to time. It can be clearly seen that there is hardly any packet loss. The reason behind it is that the real audio data is not that much bursty as well as the network parameters between nodes '0' node '1' are more than enough for such communication.

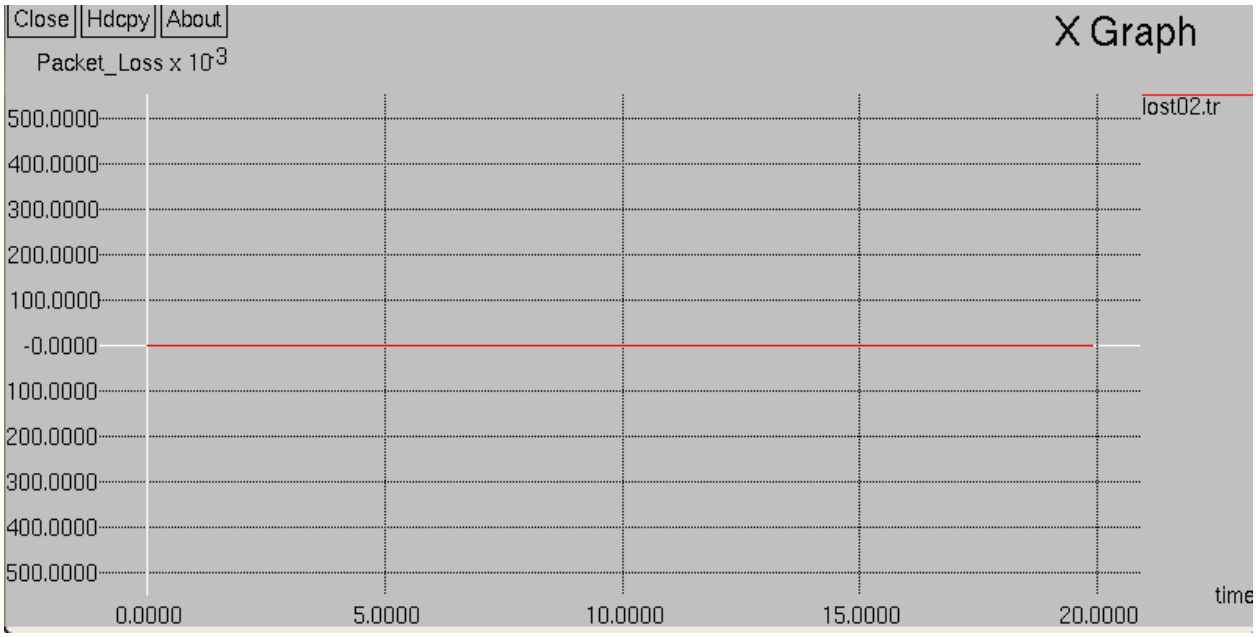

Figure-15: Packet loss for a real audio data without using token bucket shaper.

4.2.2. Simulation Results with Token Bucket Shaper: The simulation was first run for 20 seconds with the token bucket shaper attached to the node. The NAM showed the two nodes in which the data was shaped while the bandwidth window between the two nodes showed the shaped traffic. The Xgraph showed the delay, bandwidth or throughput and packet loss.

After attaching the token bucket shaper to the node '0', the real audio data was smoothened. The bursty and randomness was taken care of. The parameters used are as follows

- Bucket Size is 1024 Bytes.
- The token generation rate is 32.768k
- The queue length is 100

The data is shaped as per network agreement, which tackles the issue of packet loss and the QoS guarantee is provided.

The effect of token bucket shaper on throughput, delay and packet loss is discussed in the subsections below.

4.2.2.1. Observations for throughput: The throughput of the real audio signal with using the token bucket shaper is shaped and the randomness and burstiness is minimized. Figure-16a shows the simulation scenario in NAM. Data can be seen flowing between nodes ‘0’ and ‘1’. The Data is not random and bursty and has been shaped.

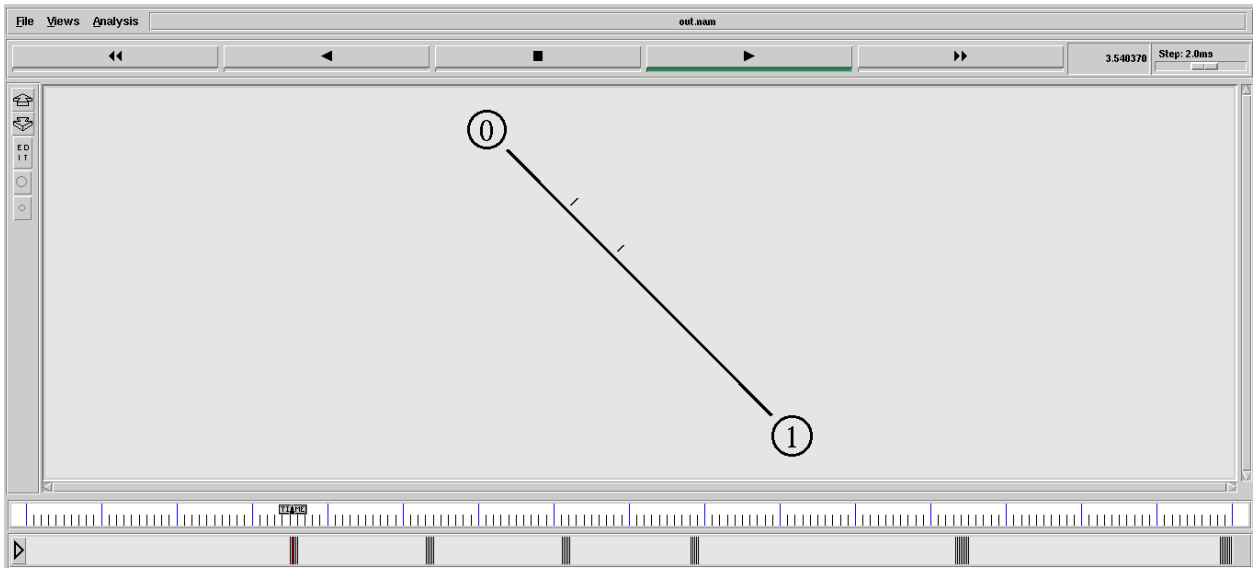

Figure-16a: The simulation scenario for real audio signal with token bucket shaper

Figure-16b shows the NAM bar which contains the shaped data. The closely packed vertical lines indicate that the real audio data has been shaped. There is a clear contrast between figure-16b and figure-13b.

Figure-16b shows the throughput of shaped real audio signal

Figure-16c shows the throughput of a shaped real audio data by using Xgraph utility. There is no significant different between figure-13c and figure-16c for the above mentioned reason i.e. the real audio data is not as bursty as MPEG4 and hence it does not violate the network agreement to a huge extent.

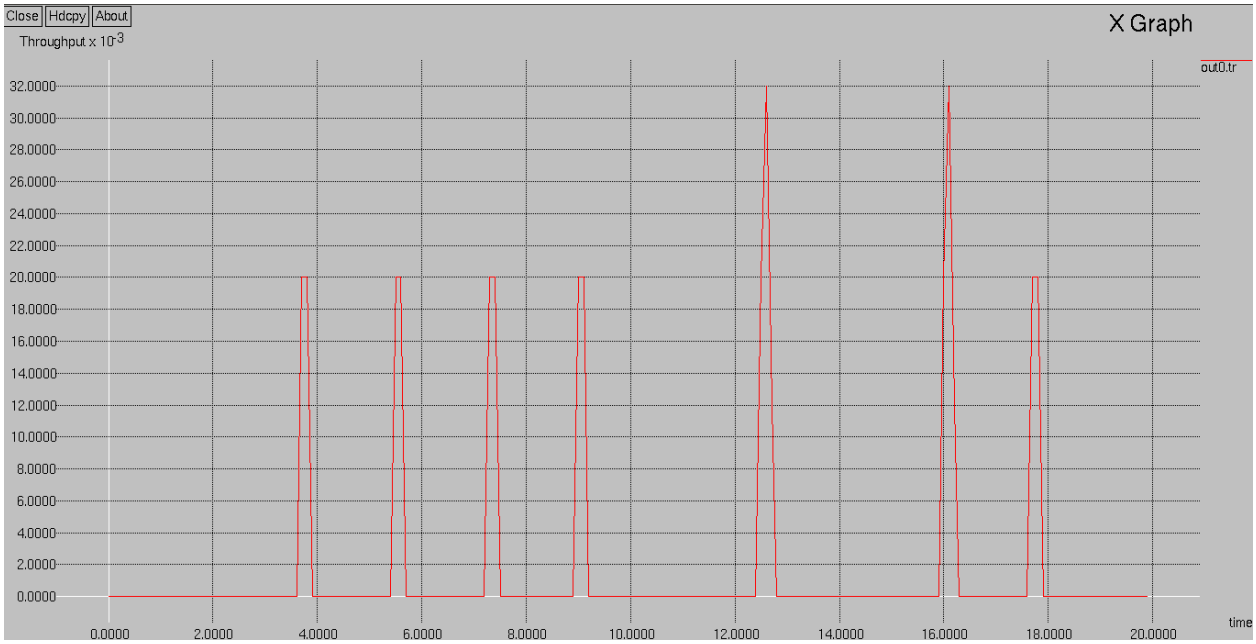

Figure-16c: The throughput of the real audio with the use of token bucket shaper.

4.2.2.2. Observation for Delay: There is some delay introduced by using the token bucket shaper for shaping the real audio data. The reason for that is understood i.e. the data waits in queue till there are sufficient tokens generated for it.

Figure-17 shows the delay which is introduced between the two nodes ‘0’ and ‘1’. It can be seen that this delay is much more compared to Figure-14 when there was no token bucket shaper used.

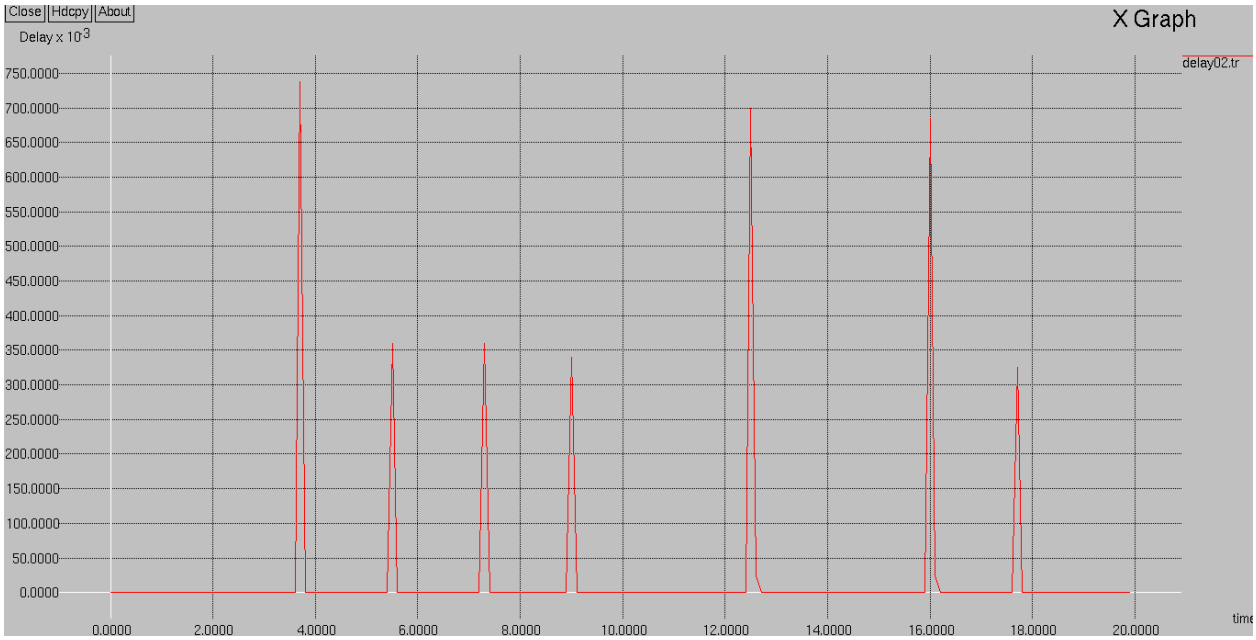

Figure-17: The delay vs time plot of shaped real audio

4.2.2.3. Observation for Packet Loss: The packet loss reduces significantly with the use of token bucket shaper. Our real audio signal initially did not have a lot of non-conformant packets which are even further reduced by the token bucket shaper so the packet loss is even more reduced.

Figure-18 shows the packet loss with respect to time for the real audio signal. The packet loss is absent which is a positive aspect.

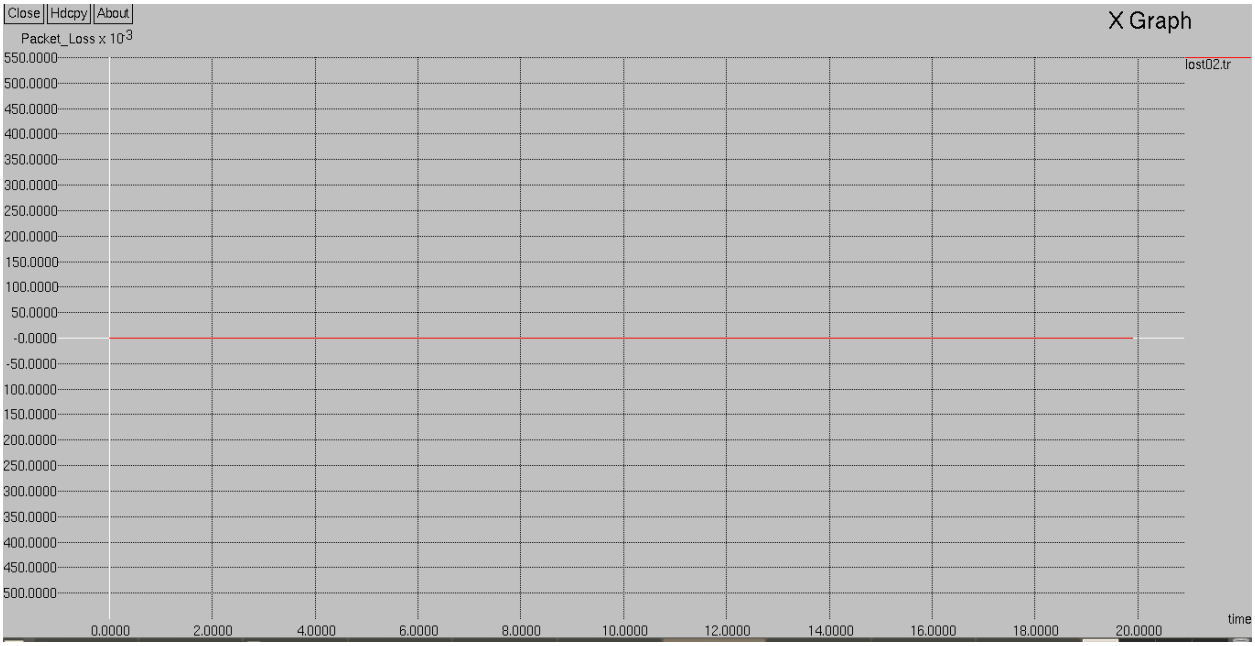

Figure-18: Packet loss for a real audio data using token bucket shaper.

4.2.3 Variation of the Token Bucket Shaper's Parameters for Real Audio data: Having seen the performance of the token bucket shaper for a real audio signal, we noticed that there is a reduction in packet loss and burstiness by using token bucket shaper. There was also some delay introduced.

There is a need for finding out the optimum token bucket shaper parameters at which the TB shaper will perform its best. It is needed to find out what will be the effect of varying various TB shaper parameters. What will happen if we tweak the values of token bucket size, the token generation rate and the queue size? This section presents the observations when the above mentioned parameters were varied in simulation.

4.2.3.1. Variation of Queue Length: In the above mentioned simulation scenario we used three different values of queue length which were 10, 100, 1000. Variation of queue length has different effects on the throughput, delay and packet loss which are presented in the subsections.

4.2.3.1.1. Effect on Throughput: Throughput for the real audio signal, for the values of 10,100 and 1000 queue length did not vary at all. However lowering the queue length to 1 caused packet loss and delay increased as well.

Chart-10 illustrates the above mentioned phenomenon.

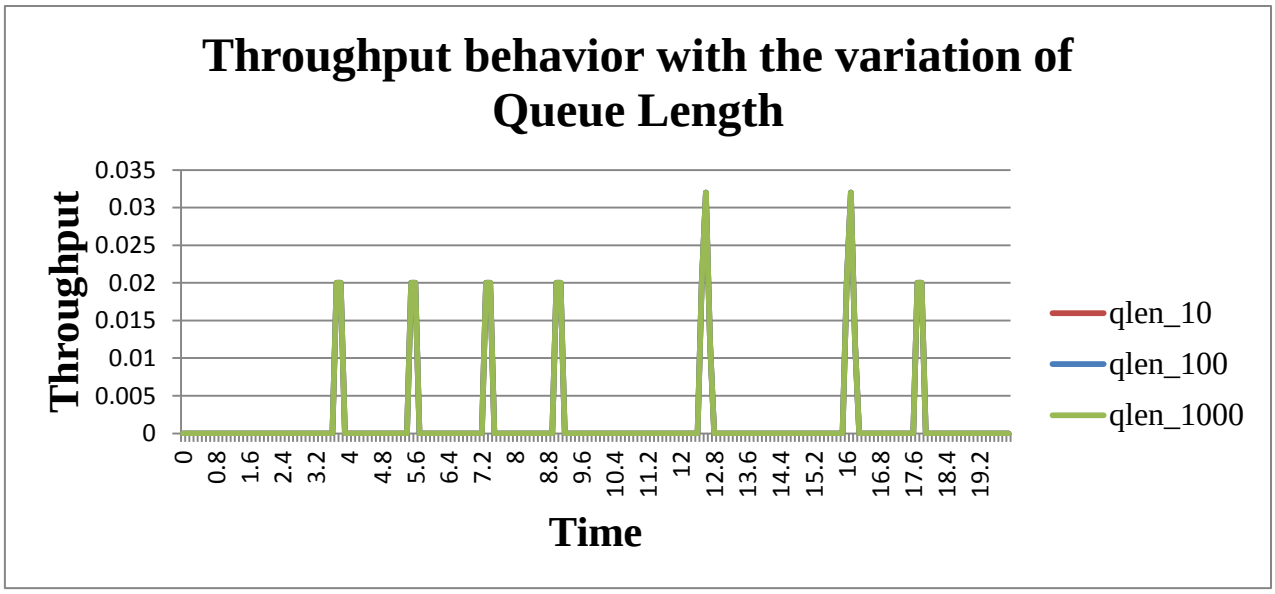

Chart-10: The effect of varying the Queue Length on the Throughput.

4.2.3.1.2. Effect on Delay: The delay was not affected at all by varying the queue length between 10, 100 and 1000.

Chart-11 presents the simulation results.

4.2.3.1.3. Effect on Packet Loss: The change in queue length from 10-1000 did not vary the packet loss at all. However decreasing the queue length to 1 causes the introduction of packet loss.

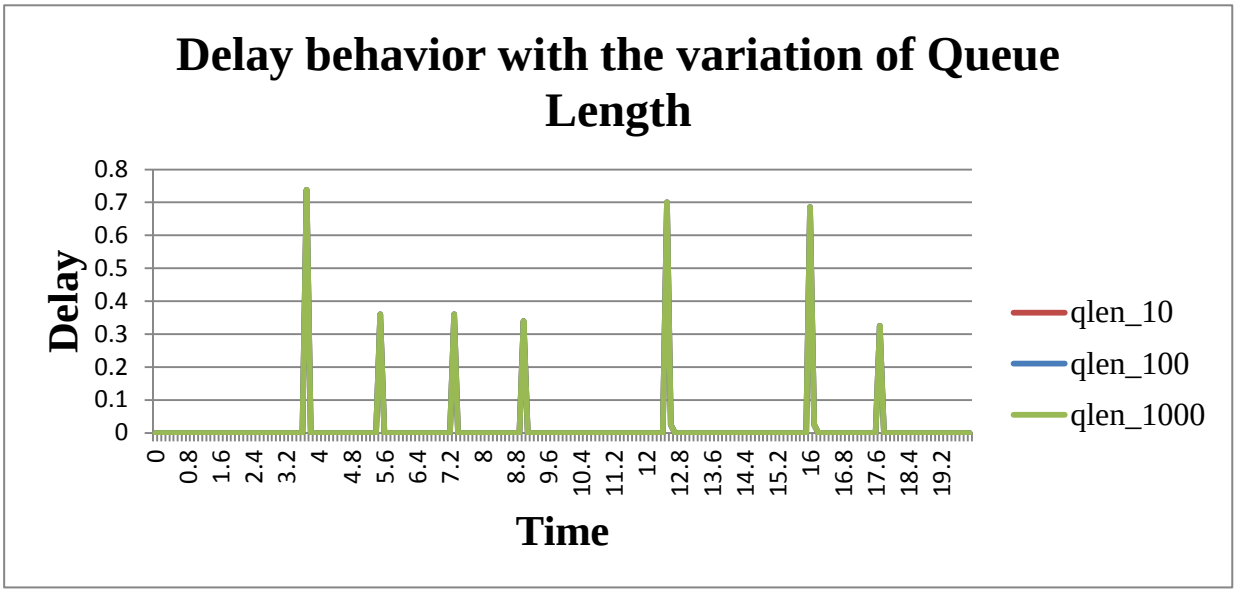

Chart-12 shows the effect of varying the queue length on Delay.

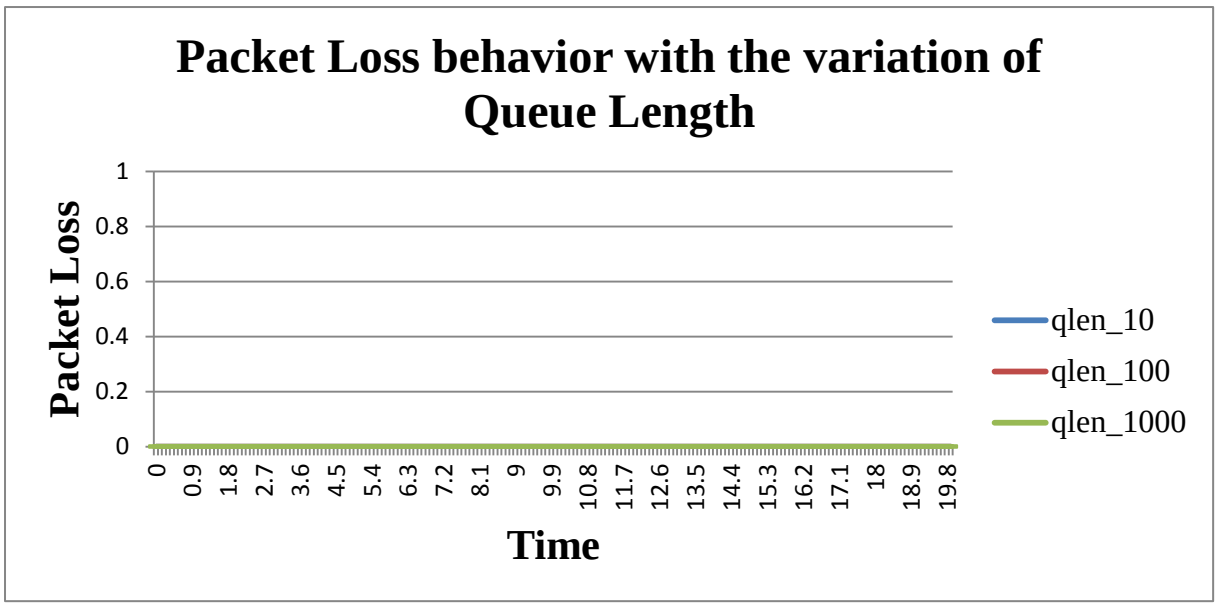

Chart-12: Effect of varying the Queue Length on Packet Loss.

4.2.3.2. Variation of Bucket Size: In the above mentioned simulation scenario we used three different values of bucket size which were 512, 1024, 2048. Variation of bucket size has different effects on the throughput, delay and packet loss which are presented in the subsections.

4.2.3.2.1. Effect on Throughput: The effect of increasing the bucket size on throughput is that burstiness increases with it. The peak value of the throughput is more for higher bucket sizes compared to lower bucket sizes.

Chart-13 shows the variation of the throughput with the bucket size.

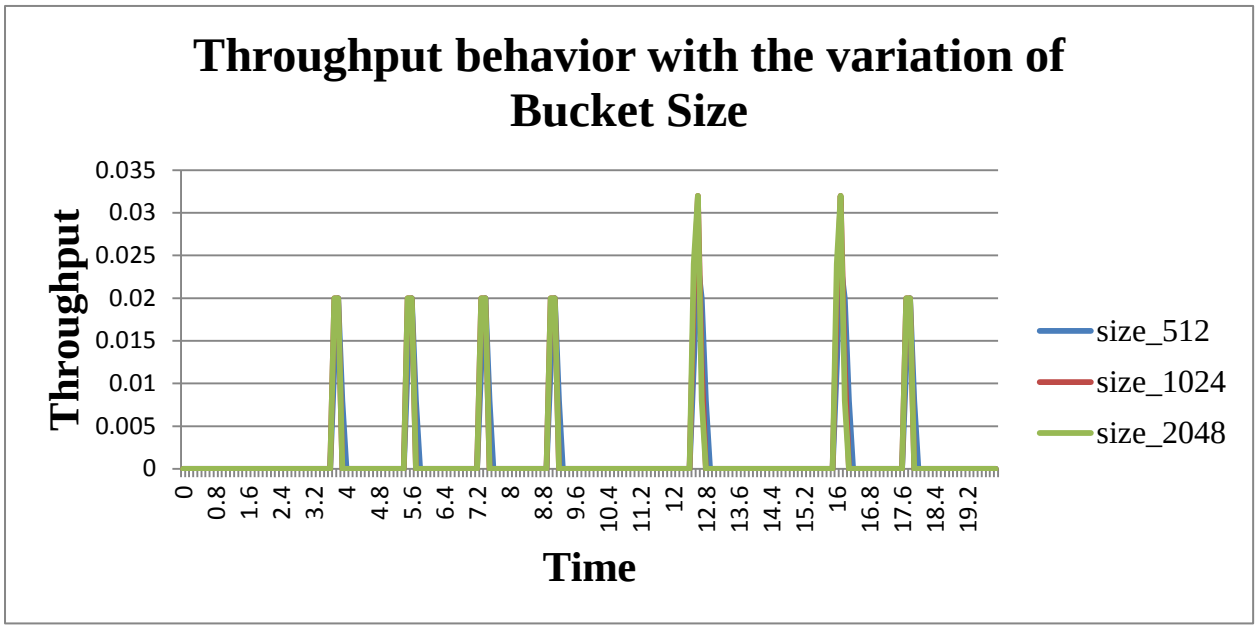

Chart-13: The variation of Throughput with the Bucket Size.

4.2.3.3.2. Effect on Delay: Increasing the bucket size decreases the delay as seen in Chart-14. It can be clearly seen that the delay time for higher bucket sizes is much less compared to the lower bucket sizes.

4.2.3.2.3. Effect on Packet Loss: Simulation showed that varying the Bucket Size between 512, 1024 and 2048 did not affect the packet loss at all. Chart-15 verifies that packet loss does not vary by changing the bucket size between 512-2048.

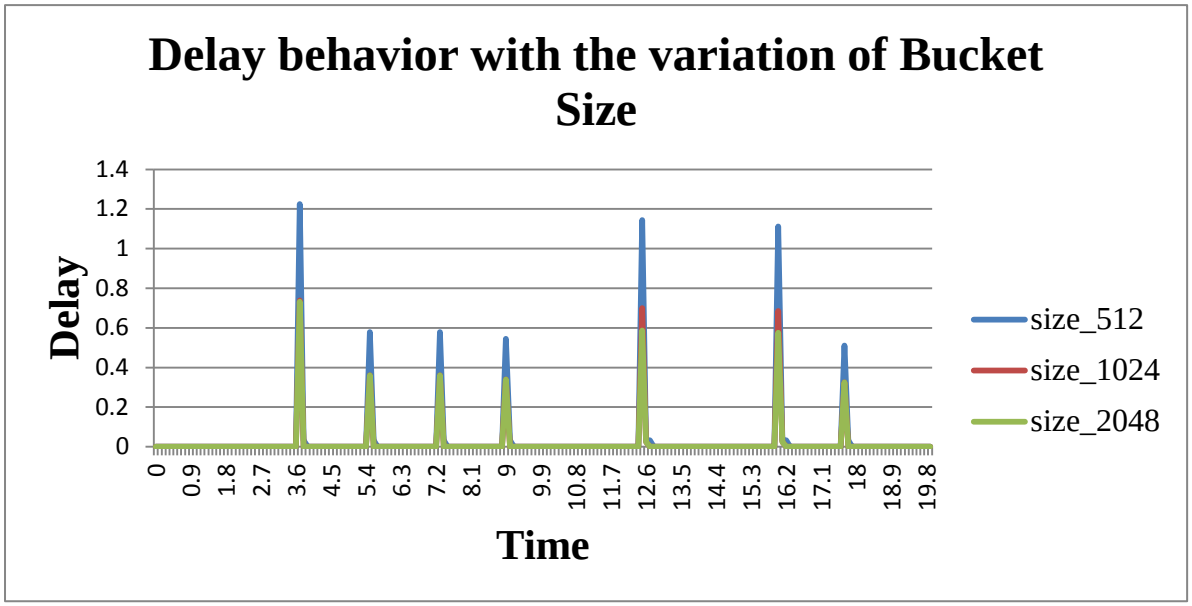

Chart-14: The variation in Delay with varying Bucket Size.

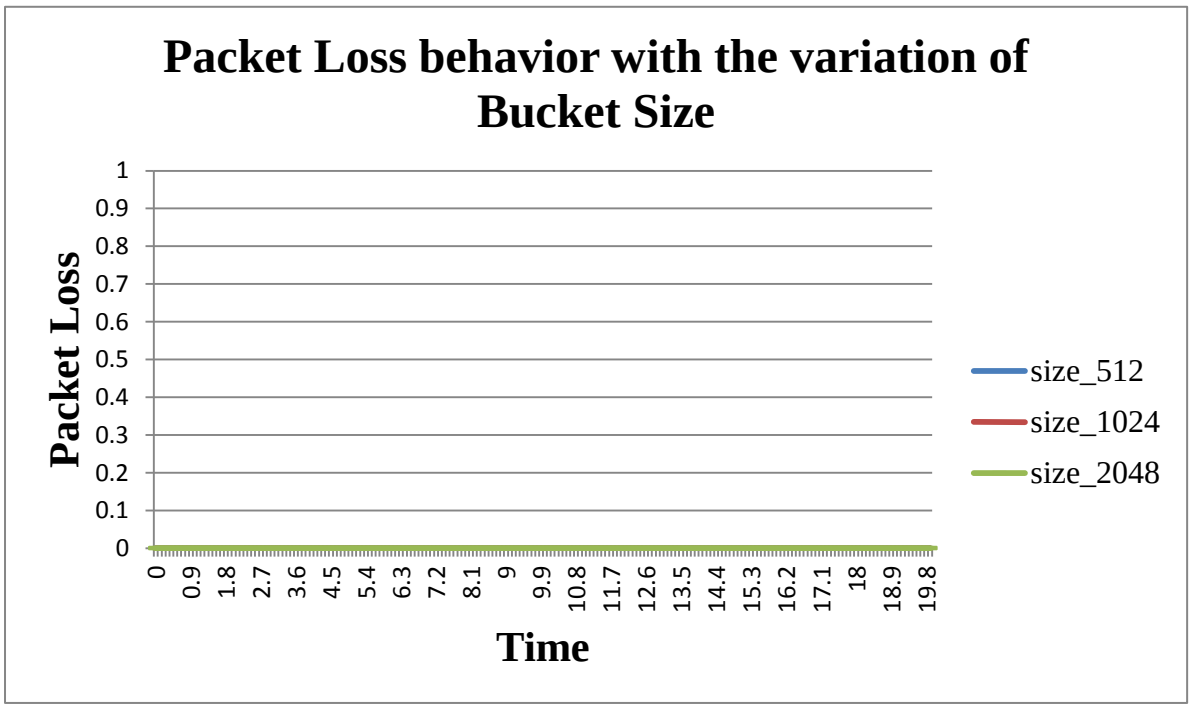

Chart-15: The variation in Packet Loss with varying Bucket Size.

4.2.3.3. Variation of Token Generation Rate: In the above mentioned simulation scenario we used three different values of token generation rate which were 16.384k, 32.768k, and 65.536k Variation of token generation rate has different effects on the throughput, delay and packet loss which are presented in the subsections.

4.2.3.3.1 Effect on Throughput: Increasing the Token Generation Rate increases the burstiness of data. Chart-16 shows the effect of varying the token generation rate on the throughput.

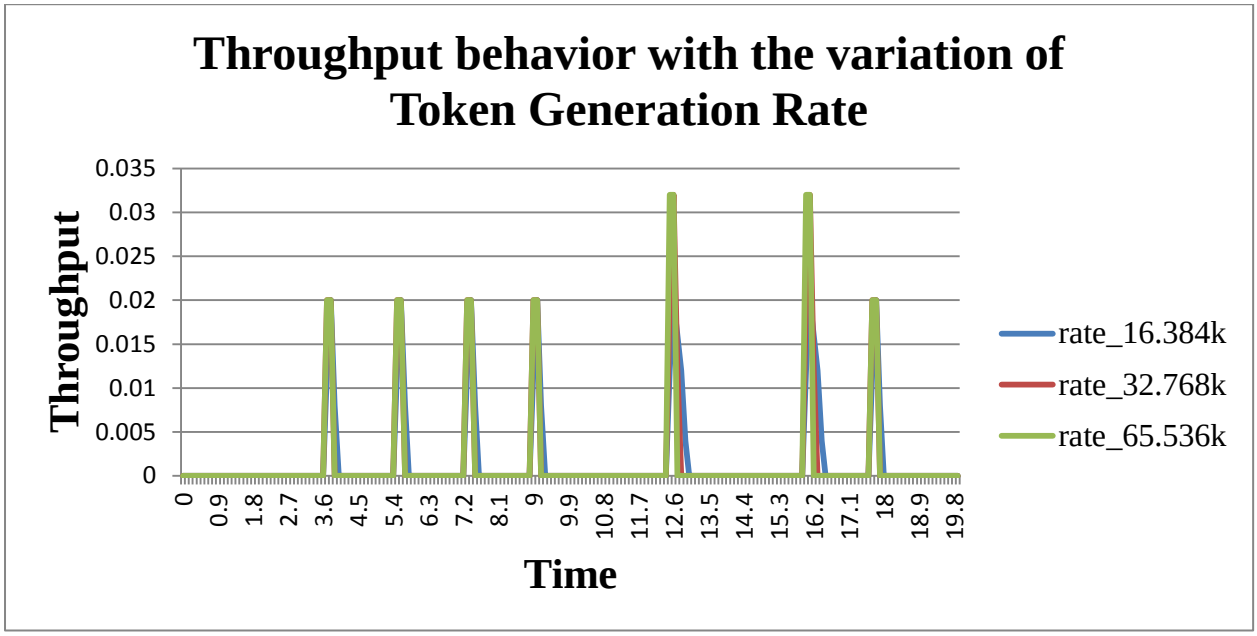

Chart-16: The effect of varying token generation rate on Throughput.

4.2.3.3.2 Effect on Delay: Increasing the token generation rate causes a decrease in the delay time and vice versa. The quicker the tokens are received, the less will be the delay.

Chart-17 reflects the above mentioned statement.

4.2.3.3.3 Effect on Packet Loss: Increasing/decreasing the token generation rate between the above mentioned limits had no effect on packet loss.

Chart-18 shows the behavior of the packet loss for the variation in the token generation rate.

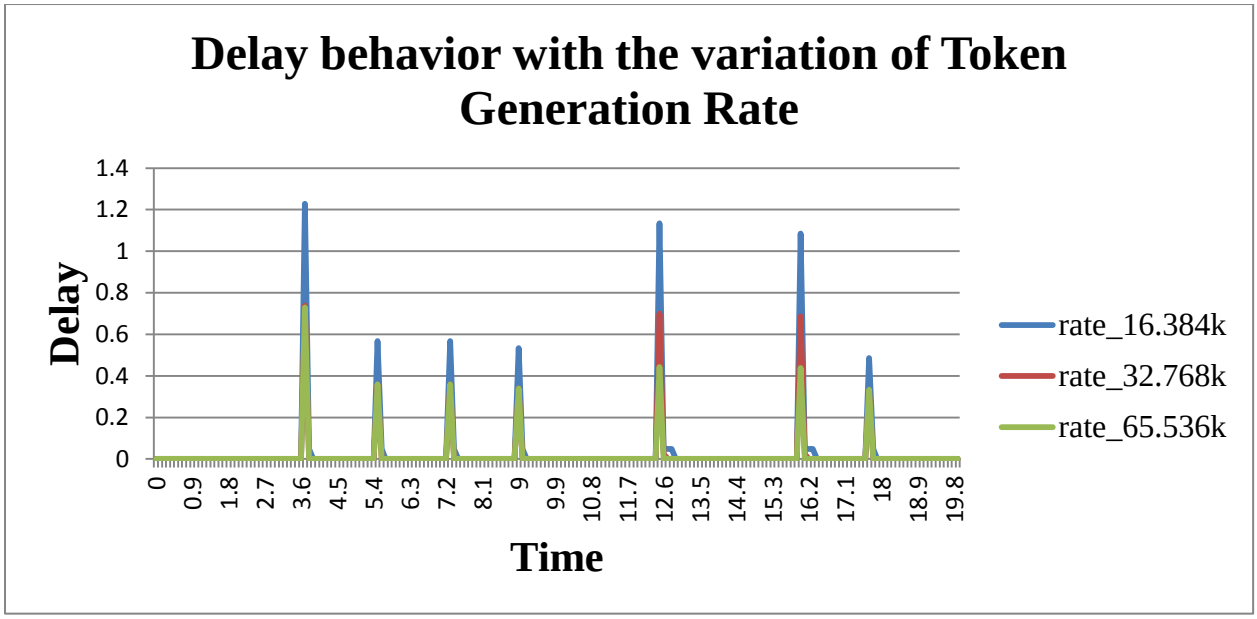

Chart-17: The effect of variation of Token generation rate on delay.

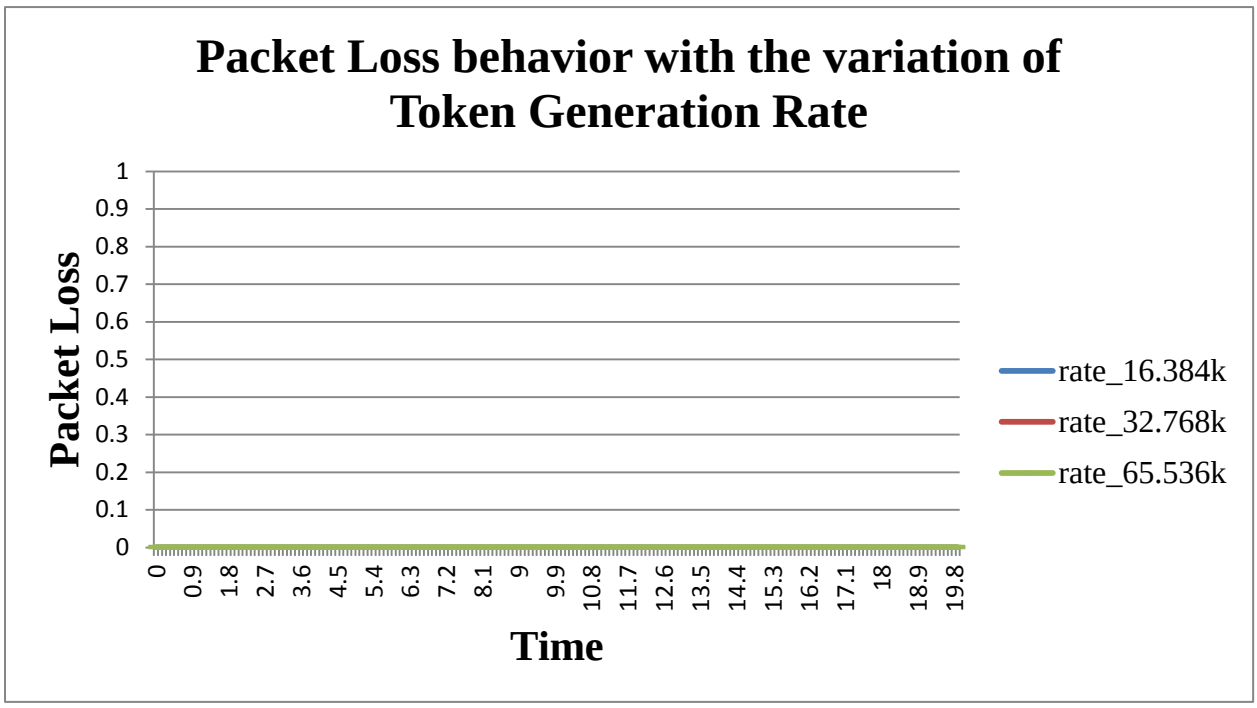

Chart-18: The effect of variation of Token generation rate on Packet Loss.

4.3. Simulation results for both MPEG4 video and Real Audio data:

The simulation environment consists of 4 nodes in which node ‘0’ acts as the MPEG4 video source node while node ‘3’ acts as the sink. Node ‘2’ acts as the real audio source. The data from node ‘0’ and node ‘1’flows into node ‘3’ through node ‘1’. The links between the nodes ‘0’, ‘1’ and ‘2’ are 2.20 Mbps with 100ms delay while the link between node ‘1’ and node ‘3’ is 0.02Mbps with 100ms delay. The UDP agent is connected to sink and data sources.

4.3.1. Simulation Results without Token Bucket Shaper: The simulation was first run for 20 seconds without any token bucket shaper so the NAM showed the simulation scenario in which the data was unshaped while the throughput window of the data sent showed the unshaped traffic. The Xgraph showed the delay, bandwidth or throughput and packet loss.

4.3.1.1 Observations for throughput: As there is the absence of the token bucket shaper at source nodes for data shaping so the throughput or packets that are received at node ‘3’ are expected to be random and bursty.

Figure-19a shows the simulation scenario in NAM. Figure-19b shows the NAM ‘bandwidth bar’, where the first set of bar indicates the unshaped real audio data. The second bar is the set of the unshaped MPEG4 video data while the third bar is the collection of MPEG4 and real audio data which goes into node 3 through node 1. It can be clearly seen that the data is random and bursty.

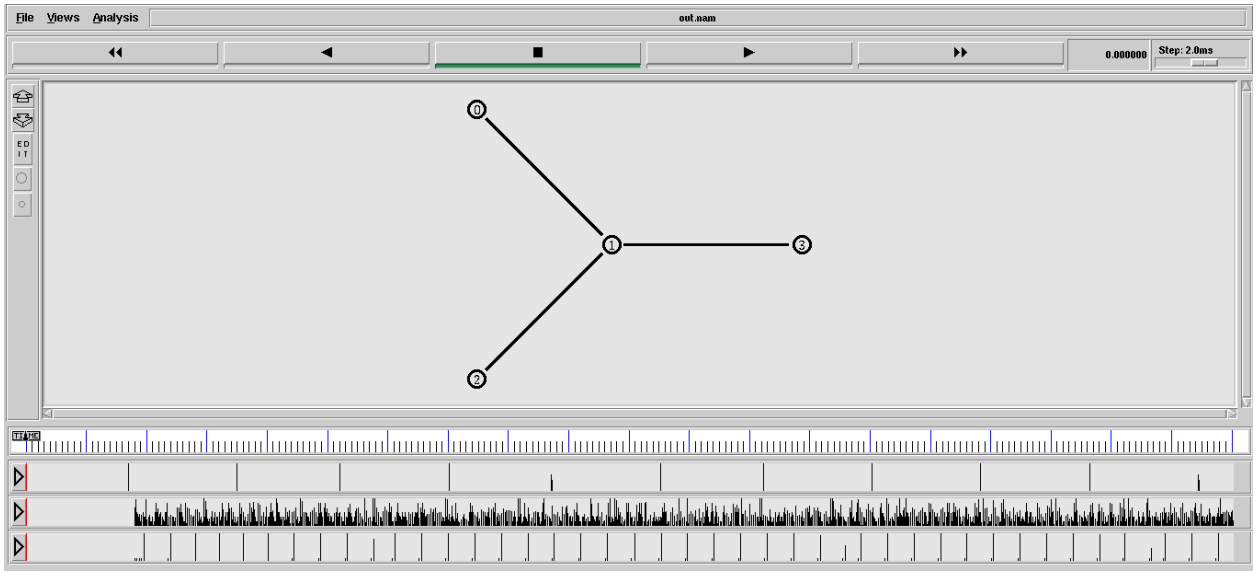

Figure-19a: Simulation Scenario for both MPEG$ and real audio data without token bucket shaper

Figure-19c shows the throughput or data received at node 3 using the Xgraph utility.

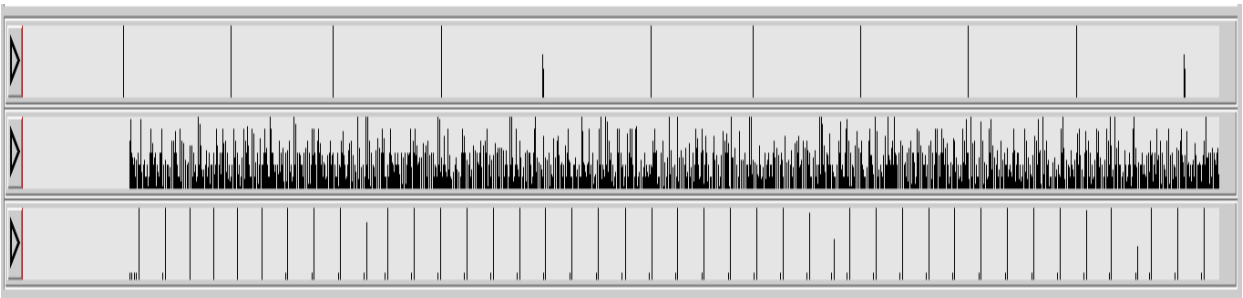

Figure-19b shows the throughput of unshaped real audio and MPEG4 data

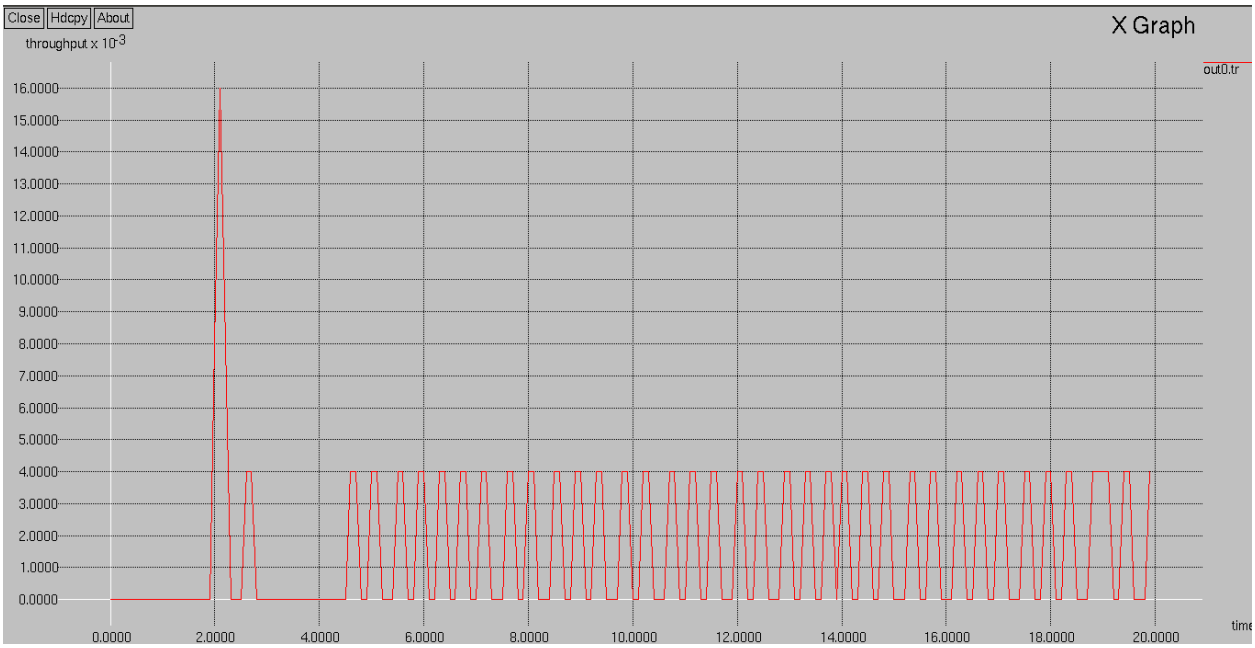

Figure-19c: The throughput of the real audio and MPEG4 video source without the use of token bucket shaper.

4.3.1.2 Observations for delay: There is some delay present without even the use of token bucket shaper.

Figure-20 shows the delay which is present in the under study scenario. It can be seen initially there is huge delay at time =2 and 4.3 seconds, while the delay reduces later on.

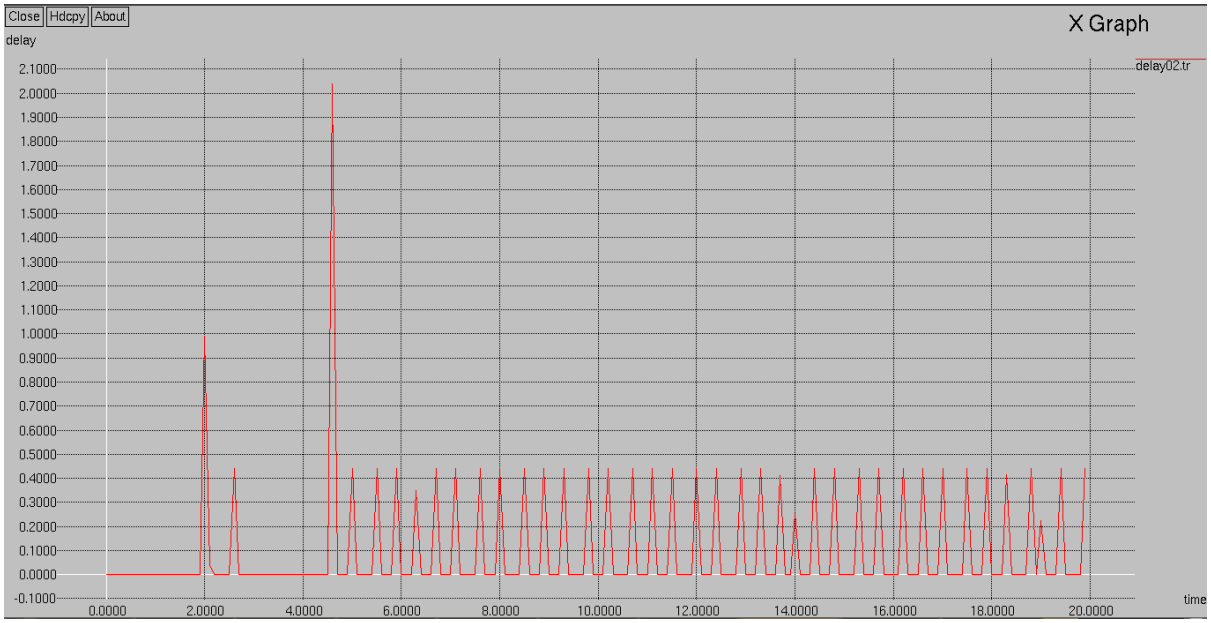

Figure-20: The delay vs. time plot of unshaped real audio and MPEG4 data.

4.3.1.3 Observations for Packet Loss: One of the major issues with the absence of token bucket shaper is that the non-conformant packets are discarded hence causing loss of packets. These packets violate the network agreement therefore they cannot be sent into the network which causes trouble.

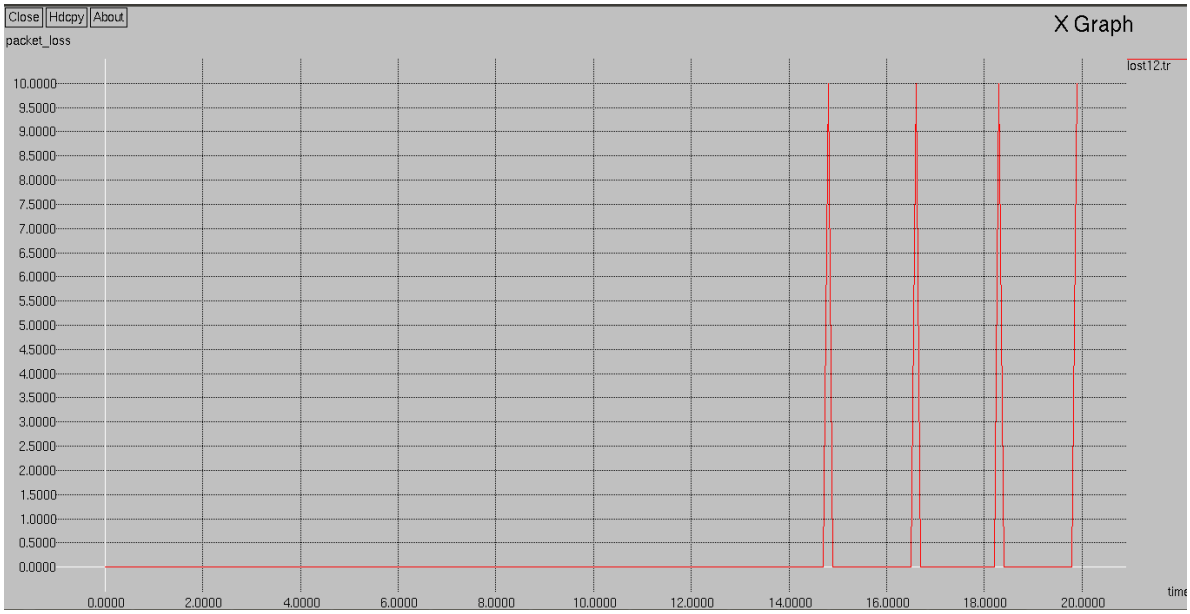

Figure-21: The Time vs. Packet Loss plot of unshaped real audio and MPEG4 data.

Figure-21 shows the packet loss. As it can be seen that there are a large number of packets which have been discarded towards the end of simulation hence for audio, video communications; the loss of these packets can disrupt the whole communication.

4.3.2. Simulation Results with Token Bucket Shaper: The simulation was first run for 20 seconds with the token bucket shaper attached to the node. The NAM showed the nodes in which the data was shaped while the bandwidth windows between the nodes showed the shaped traffic. The Xgraph showed the delay, bandwidth or throughput and packet loss.

After attaching the token bucket shaper to the node '2', the real audio data was smoothened while the token bucket shaper at node '0' shaped the MPEG4 video. The bursty and randomness was taken care of. The parameters used for the token bucket shapers are as follows

- Bucket Size is 1024 Bytes.
- The token generation rate is 32.768k
- The queue length is 100

The data is shaped as per network agreement, which tackles the issue of packet loss and the QoS guarantee is provided. The effect of token bucket shaper on throughput, delay and packet loss is discussed in the subsections below.

4.3.2.1. Observation for Throughput: The use of the Token bucket shaper regulates and smoothens the data sent by node '0' and node '2' to node '3'via node '1'. The traffic is shaped as per network parameters.

Figure-22a presents the NAM output showing the simulation scenario with the use of token bucket shapers at node '0' and node '2'. Figure-22b shows the bandwidth bar in NAM which shows the shaped data. Figure-22c shows the throughput or data received using Xgraph.

4.3.2.2. Observation for Delay: The introduction of the token bucket shaper causes some delay as the token generation for packets takes some time. The delay can be seen in figure-23 which on average is more than the delay shown in figure-20.

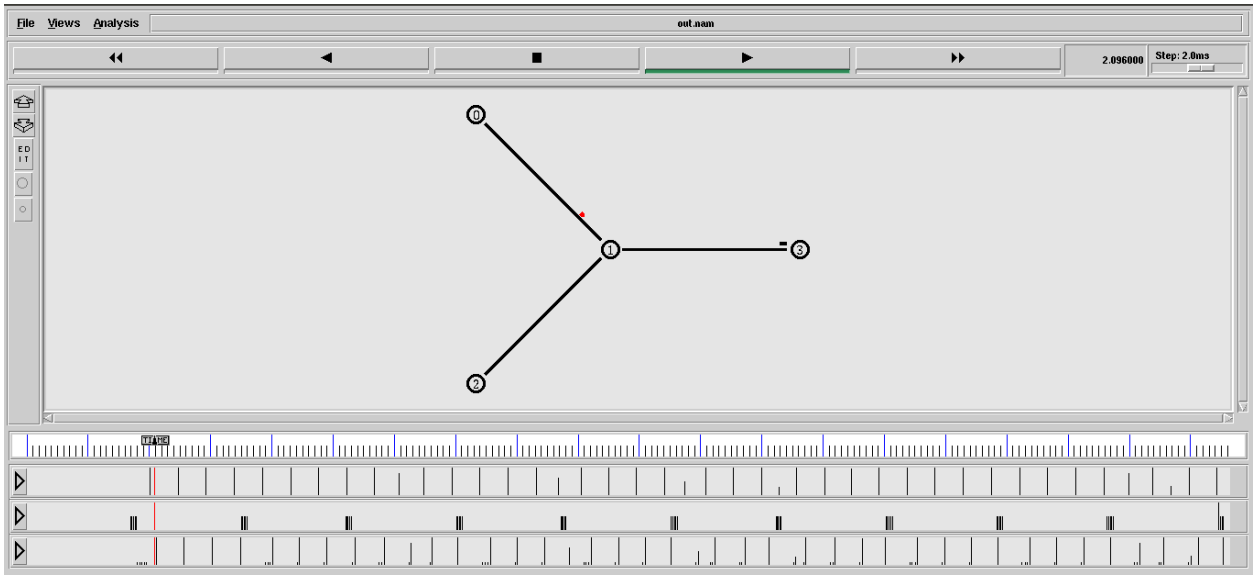

Figure-22a: Simulation Scenario for both MPEG$ and real audio data without token bucket shaper

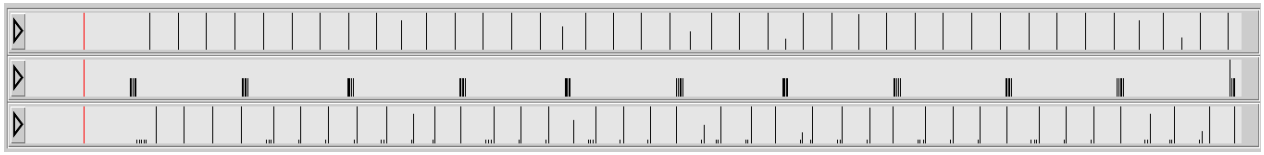

Figure-22b shows the throughput of shaped real audio and MPEG4 data

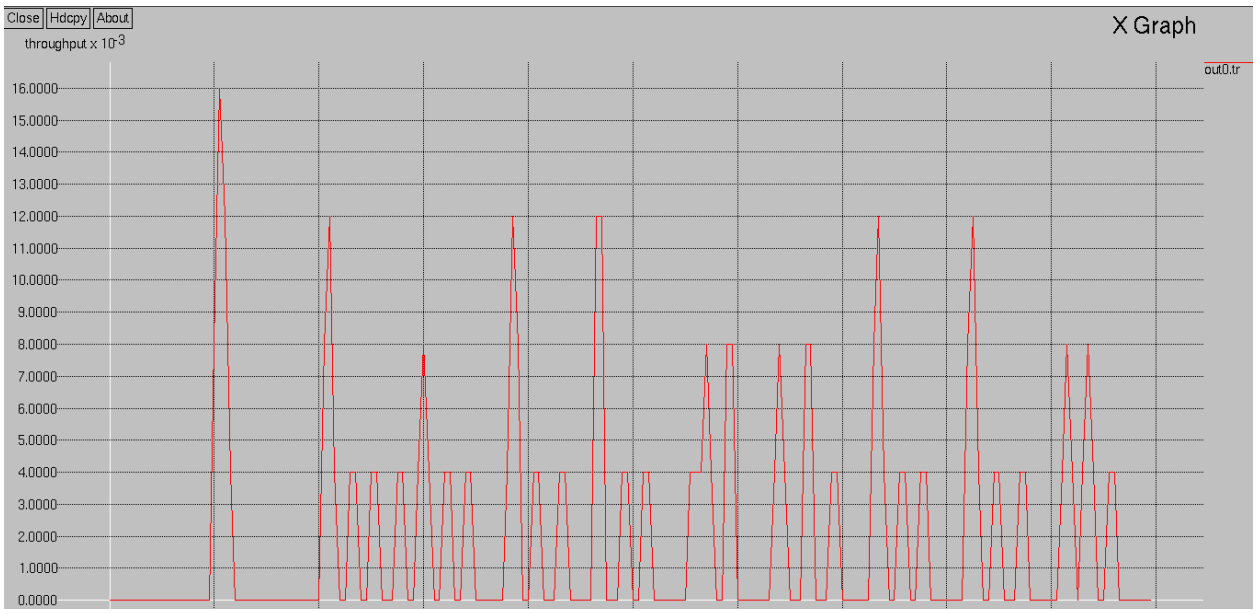

Figure-22c: The throughput of the real audio and MPEG4 video source with the use of token bucket shaper.

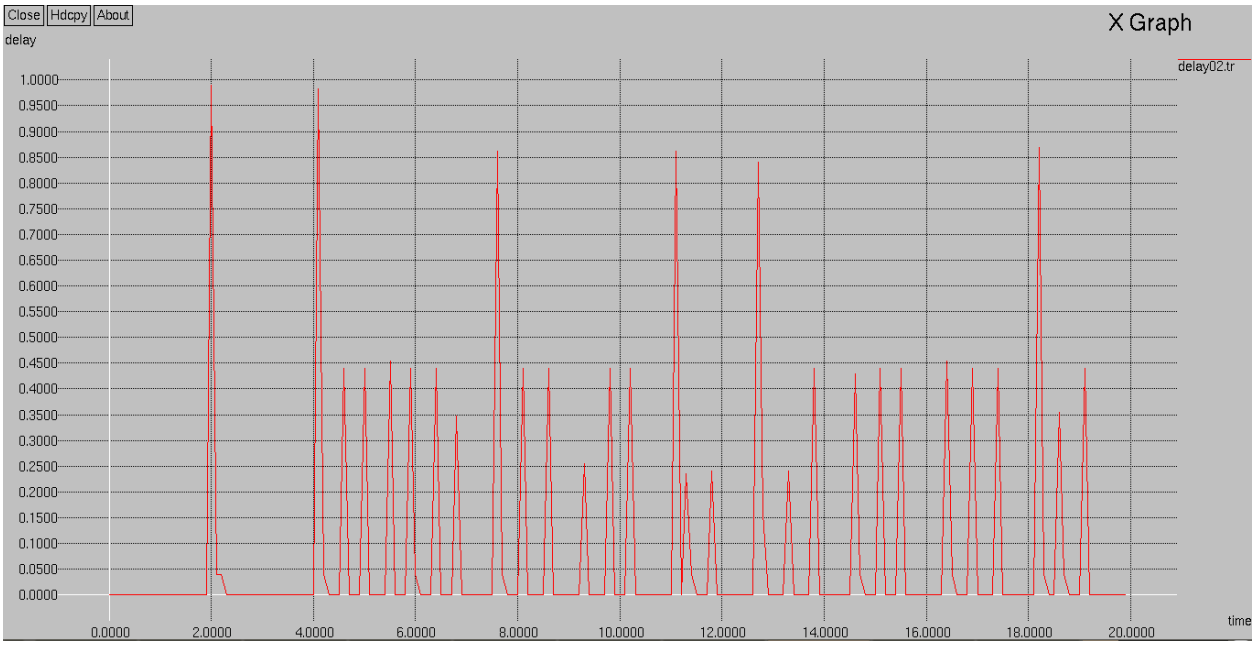

Figure-23: The delay vs. time plot of shaped real audio and MPEG4 data.

4.3.2.3. Observations for Packet Loss: The use of token bucket shaper helps us get rid of the packet loss as shown in figure-24. This is one of the most important advantages of the token bucket shaper. If we compare figure-21 and figure-24 we see a total contrast between the performance. The performance got much better and improved.

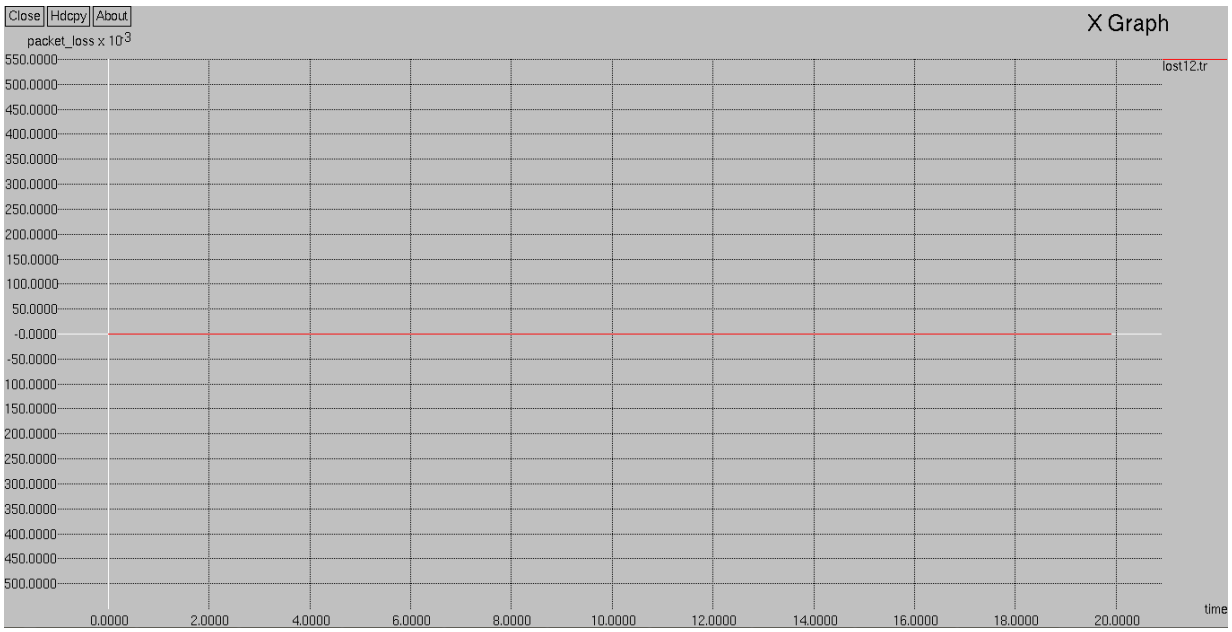

Figure-24: Packet loss for real audio and MPEG4 video data using token bucket shaper.

4.3.3 Variation of the Token Bucket Shaper's Parameters for Real Audio and MPEG4 Data: Since independent Token Bucket Shapers are used so varying the Token Bucket Shaper parameters will cause the same effects as mentioned in Sections 4.1.3 and 4.2.3. The behavior mentioned there will be replicated here as well.

Chapter 5

Conclusion: Future Plans

Modern age has witnessed a drastic increase in the number of technology users. The communication between these users increases the demand for better networking and provision of QoS guarantee to each and every user. There is a bilateral agreement between the user and network or service provider which has to be taken care of at all cost for the provision of the best services at affordable cost to each and every user. To make sure that the network agreement is not violated (which will cause packet loss), the network has policing algorithm while on the user end there is need for traffic shaping algorithm. Token Bucket algorithm is a traffic shaping algorithm on which the token bucket shaper works. The token shaper shapes the traffic as per the network parameters and provides smooth throughput (unlike the unregulated traffic which is bursty and random). The use of token bucket shaper with MPEG4 video, real audio signal and the combination of two showed that token bucket shaper can efficiently regulate the data sent which helps us get rid of packet loss.

The use of MPEG4 and real audio signal trace files in NS-2 replicated the real world MPEG4 and real audio signal and provided us with satisfactory results. The packet loss was minimized while the traffic was smoothened and then we reduced the delay to the optimum value. The results shown in chapter 4 provided us with the proof that token bucket shaper is an efficient tool in handling the packet loss and regulating the user traffic. The randomness and burstiness that is found in the real world data such as MPEG4 video and real audio data, is dealt by the token bucket shaper efficiently.

The ability to tweak the values of the token bucket shaper's parameters is an added bonus. It is extremely usefully and obtaining an optimum performing token bucket shaper. Varying the token bucket size, token generation rate and queue length provides us with different results, which based on the user's choice of trade-offs; will perform at its optimum ability.

This thesis relied on real audio and MPEG4 data which was static i.e. we used the trace files of various audio and MPEG4 video signals. In future, there is need to analyze the data coming from VoIP source which means the real time data from VoIP source, Microphone and other input sources directly fed into NS-2 for analysis. This requires the introduction of new modules into NS-2 which is not available yet. However performing

analysis of this real time data will give us even better glimpse of how the token bucket shaper can shape the data and provide us the QoS guarantee in converged networks which support audio and video data. Improving the token bucket shaper will enhance the network abilities as the network will perform better without the need of dropping packets.

In an age where the increasing the network size is not an issue compared to the issue of QoS guarantee, there is a need for improving the traffic shaping since it provides the best possible QoS guarantee.

APPENDIX A

Code For MPEG4 Video Data

```
set ns [new Simulator]

set f [open out.tr w]

set f0 [open out0.tr w]

set f5 [open lost02.tr w]

set f8 [open delay02.tr w]

set n0 [$ns node]

set n1 [$ns node]

$ns trace-all $f

set nf [open out.nam w]

$ns namtrace-all $nf

set trace_flow 1

$ns color 0 red

$ns color 1 black

$ns duplex-link $n0 $n1 2.20Mbps 100ms SFQ

$ns duplex-link-op $n0 $n1 orient right-down

set udp0 [new Agent/UDP]

$ns attach-agent $n0 $udp0

set vdo [new Application/Traffic/MPEG4]

$vdo set initialSeed_ 0.4

$vdo set rateFactor_ 5

$vdo attach-agent $udp0
```

```
set tbf [new TBF]

$tbf set bucket_ 1024

$tbf set rate_ 32.768k

$tbf set qlen_  1000

$ns attach-tbf-agent $n0 $udp0 $tbf

set sink0 [new Agent/LossMonitor]

$ns attach-agent $n1 $sink0

$ns connect $udp0 $sink0

set holdtime 0

set holdseq 0

set holdtime1 0

set holdseq1 0

set holdtime2 0

set holdseq2 0

set holdtime3 0

set holdseq3 0

set holdrate1 0

set holdrate2 0

set holdrate3 0

set holdrate4 0

proc record {} {
```

```
   global f0 sink0 sink sink2 sink3 sink4 f0 f1 f2 f3 f4 f5 f6 f7 holdtime holdseq holdtime1
holdseq1 holdtime2 holdseq2 holdtime3 holdseq3 f8 f9 f10 f11 holdrate1 holdrate2 holdrate3
holdrate4

        set ns [Simulator instance]

         set time 0.1

        #How many bytes have been received by the traffic sinks?

        set bw0 [$sink0 set bytes_]


       set bw5 [$sink0 set nlost_]

        set bw8 [$sink0 set lastPktTime_]

        set bw9 [$sink0 set npkts_]

        #Get the current time

        set now [$ns now]

          puts $f0 "$now [expr (($bw0+$holdrate1)*8)/(2*$time*1000000)]"

puts $f5 "$now [expr $bw5/$time]"

if { $bw9 > $holdseq } {

              puts $f8 "$now [expr ($bw8 - $holdtime)/($bw9 - $holdseq)]"

        } else {

              puts $f8 "$now [expr ($bw9 - $holdseq)]"

        }

        $sink0 set bytes_ 0

        $sink0 set nlost_ 0

set holdtime $bw8
```

```
        set holdseq $bw9
        set  holdrate1 $bw0
        #Re-schedule the procedure
        $ns at [expr $now+$time] "record"
}
$ns at 0.0 "record"
$ns at 1.8 "$vdo start"
$ns at 20.0 "stop"
proc stop {} {

        global ns tracefd f0 f1 f2 f3 f4 f5 f6 f7 f8 f9 f10 f11
          close $f0
        close $f5
        close $f8
        # Plot Recorded Statistics
        exec xgraph out0.tr  -x time -y Throughput -geometry 800x400 &
      exec xgraph lost02.tr -x time -y Packet_loss -geometry 800x400 &
        exec xgraph delay02.tr -x time -y Delay  -geometry 800x400 &
       exec nam out.nam
        # Reset Trace File
        $ns flush-trace
        close $tracefd
        exit 0
```

}

$ns run

APPENDIX B

Code for Real Audio Data

```
set ns [new Simulator]

set f [open out.tr w]

set f0 [open out0.tr w]

set f5 [open lost02.tr w]

set f8 [open delay02.tr w]


set n0 [$ns node]

set n1 [$ns node]


$ns trace-all $f

set nf [open out.nam w]

$ns namtrace-all $nf

set trace_flow 1

$ns color 0 red

$ns color 1 black


$ns duplex-link $n0 $n1 .020Mbps 100ms SFQ


$ns duplex-link-op $n0 $n1 orient right-down


set realaudio [new Application/Traffic/RealAudio]
```

```
  $realaudio set packetSize_ 100


  $realaudio set burst_time_ 0.05ms
  $realaudio set idle_time_ 1800ms
set udp0 [new Agent/UDP]
$udp0 set fid_ 1
$udp0 set rate_ 32.768k
$udp0 set bucket_ 1024

$realaudio attach-agent $udp0

set tbf [new TBF]
$tbf set bucket_ 1024
$tbf set rate_ 32.768k
$tbf set qlen_  100
$ns attach-tbf-agent $n0 $udp0 $tbf
set sink0 [new Agent/LossMonitor]
$ns attach-agent $n1 $sink0
$ns connect $udp0 $sink0
set holdtime 0
set holdseq 0

set holdtime1 0
```

```
set holdseq1 0


set holdtime2 0

set holdseq2 0


set holdtime3 0

set holdseq3 0


set holdrate1 0

set holdrate2 0


set holdrate3 0

set holdrate4 0



proc record {} {


  global f0 sink0 sink sink2 sink3 sink4 f0 f1 f2 f3 f4 f5 f6 f7 holdtime holdseq holdtime1 holdseq1 holdtime2 holdseq2 holdtime3 holdseq3 f8 f9 f10 f11 holdrate1 holdrate2 holdrate3 holdrate4

      set ns [Simulator instance]

       set time 0.1

      set bw0 [$sink0 set bytes_]
```

```
        set bw5 [$sink0 set nlost_]
    set bw8 [$sink0 set lastPktTime_]
    set bw9 [$sink0 set npkts_]
    #Get the current time
    set now [$ns now]
        puts $f0 "$now [expr (($bw0+$holdrate1)*8)/(2*$time*1000000)]"
       #Reset the bytes_ values on the traffic sinks
puts $f5 "$now [expr $bw5/$time]"
if { $bw9 > $holdseq } {
        puts $f8 "$now [expr ($bw8 - $holdtime)/($bw9 - $holdseq)]"
    } else {
        puts $f8 "$now [expr ($bw9 - $holdseq)]"
    }
    $sink0 set bytes_ 0
    $sink0 set nlost_ 0
set holdtime $bw8
    set holdseq $bw9

    set  holdrate1 $bw0
    #Re-schedule the procedure
    $ns at [expr $now+$time] "record"
}
$ns at 0.0 "record"
```

```
$ns at 1.8 "$realaudio start"

$ns at 20.0 "stop"

proc stop {} {

    global ns tracefd f0 f1 f2 f3 f4 f5 f6 f7 f8 f9 f10 f11




    # Close Trace Files


    close $f0

    close $f5

    close $f8


    # Plot Recorded Statistics

    exec xgraph out0.tr -x time -y Throughput -geometry 800x400 &

    exec xgraph lost02.tr -x time -y Packet_Loss -geometry 800x400 &

    exec xgraph delay02.tr -x time -y Delay  -geometry 800x400 &

    exec nam out.nam

    # Reset Trace File

    $ns flush-trace

    close $tracefd

    exit 0
```

}

$ns run

APPENDIX C

Code for Both MPEG4 video and Real Audio Data

```
set ns [new Simulator]

set f [open out.tr w]

set f0 [open out0.tr w]


set f4 [open lost02.tr w]

set f5 [open lost12.tr w]


set f8 [open delay02.tr w]


set n0 [$ns node]

set n1 [$ns node]

set n2 [$ns node]

set n3 [$ns node]


$ns trace-all $f

set nf [open out.nam w]

$ns namtrace-all $nf

set trace_flow 1

$ns color 0 red

$ns color 1 black

$ns color 2 blue
```

```
$ns color 3 purple

$ns duplex-link $n0 $n1 2.20Mbps 100ms SFQ

$ns duplex-link $n2 $n1 2.20Mbps 100ms SFQ

$ns duplex-link $n1 $n3 .020Mbps 100ms SFQ

$ns duplex-link-op $n0 $n1 orient right-down

$ns duplex-link-op $n2 $n1 orient right-up

$ns duplex-link-op $n1 $n3 orient right

set udp0 [new Agent/UDP]

$ns attach-agent $n0 $udp0

set vdo [new Application/Traffic/MPEG4]

$vdo set initialSeed_ 0.4

$vdo set rateFactor_ 5

$vdo attach-agent $udp0

set tbf [new TBF]

$tbf set bucket_ 1024

$tbf set rate_ 32.768k

$tbf set qlen_  100

$ns attach-tbf-agent $n0 $udp0 $tbf
```

```
set sink0 [new Agent/LossMonitor]
$ns attach-agent $n3 $sink0
$ns connect $udp0 $sink0

set realaudio [new Application/Traffic/RealAudio]
  $realaudio set packetSize_ 100

  $realaudio set burst_time_ 0.05ms
  $realaudio set idle_time_ 1800ms
set a2 [new Agent/UDP]
$a2 set fid_ 1
$a2 set rate_ 32.768k
$a2 set bucket_ 1024
$realaudio attach-agent $a2
set tbf1 [new TBF]
$tbf1 set bucket_ [$a2 set bucket_]
$tbf1 set rate_ [$a2 set rate_]
$tbf1 set qlen_  100
$ns attach-tbf-agent $n2 $a2 $tbf1
set sink0 [new Agent/LossMonitor]
$ns attach-agent $n3 $sink0
$ns connect $a2 $sink0
```

```
set holdtime 0

set holdseq 0


set holdtime1 0

set holdseq1 0


set holdtime2 0

set holdseq2 0


set holdtime3 0

set holdseq3 0


set holdrate1 0

set holdrate2 0


set holdrate3 0

set holdrate4 0

proc record {} {

    global f0 sink0 sink sink2 sink3 sink4 f0 f1 f2 f3 f4 f5 f6 f7 holdtime holdseq holdtime1
holdseq1 holdtime2 holdseq2 holdtime3 holdseq3 f8 f9 f10 f11 holdrate1 holdrate2 holdrate3
holdrate4

    set ns [Simulator instance]
```

```
   set time 0.1
set bw0 [$sink0 set bytes_]
set bw5 [$sink0 set nlost_]
set bw8 [$sink0 set lastPktTime_]
set bw9 [$sink0 set npkts_]
set now [$ns now]

# Record Bit Rate in Trace Files

puts $f0 "$now [expr (($bw0+$holdrate1)*8)/(2*$time*1000000)]"
# Record Packet Loss Rate in File
puts $f5 "$now [expr $bw5/$time]"
# Record Packet Delay in File

if { $bw9 > $holdseq } {
        puts $f8 "$now [expr ($bw8 - $holdtime)/($bw9 - $holdseq)]"
} else {
        puts $f8 "$now [expr ($bw9 - $holdseq)]"
}

# Reset Variables
$sink0 set bytes_ 0
$sink0 set nlost_ 0
```

```
        set holdtime $bw8
        set holdseq $bw9
        set  holdrate1 $bw0
         $ns at [expr $now+$time] "record”

}

$ns at 0.0 "record"
$ns at 0.0 "$realaudio start"
$ns at 1.8 "$vdo start"
$ns at 20.0 "$realaudio stop"
$ns at 20.0 "stop"

proc stop {} {

        global ns_ tracefd f0 f1 f2 f3 f4 f5 f6 f7 f8 f9 f10 f11
       # Close Trace Files
        close $f0
        close $f5
        close $f8\
                exec xgraph out0.tr -x time -y throughput -geometry 800x400 &
        exec xgraph lost12.tr -x time -y packet_loss -geometry 800x400 &
```

```
        exec xgraph delay02.tr -x time -y delay   -geometry 800x400 &
       exec nam out.nam
       # Reset Trace File
        $ns_ flush-trace
        close $tracefd
        exit 0
}
$ns run
```

References:

1. T.H. Lee "Correlated Token Bucket Shapers for Multiple Traffic Classes."
2. Nguyen Hong Van "Mobile Agent Approach to Congestion Controlling Heterogeneous Networks."
3. Iman Shames, Nima Najmaei, Mohammad Zamani, A. A. Safavi "A New Intelligent Traffic Shaper for High Speed Networks".
4. Wilfried N. Gansterer, Helmut Hlavacs, Michael Ilger, Peter Lechner, J¨urgen Strauß
   "Token Buckets For Outgoing Spam Prevention".
5. David Perez, Jose Luis Valenzuela "Performance of a Token Bucket traffic shaper ona real IEEE 802.11 test-bed".
6. Partha Kanuparthy, Constantine Dovrolis "Endtoend Detection of ISP Traffic Shaping using Active and Passive Methods."
7. J.L. Valenzuela, A. Monleon, I. San Esteban, M. Portoles, O.Sallent "A Hierarchical Token Bucket Algorithm to Enhance QoS in IEEE 802.11: Proposal, Implementation and Evaluation."
8. Jianxin Liao, Jinzhu Wang, Tonghong Li, Jingyu Wang, Xiaomin Zhu "A token-bucket based notification traffic control mechanism for IMS presence service".
9. Puqi Perry Tang and Tsung-Yuan Charles Tai "Network Traffic Characterization Using Token Bucket Model".
10. Tsern-Huei Lee and Yaw-Wen Kuo "An Efficient Admission Criterion of Traffic-Shaped Rate Monotonic Scheduler for VBR Traffic".
11. R. Bruno, R. G. Garroppo and S. Giordano "Estimation of Token Bucket Parameters of VoIP Traffic."
12. Mohammad F. Alam, Mohammed Atiquzzaman, Mohammad A. Karim "Effects of Source Traffic Shaping on MPEG Video Transmission over Next Generation IP Networks"
13. Ns Manual "The VINT Project "A Collaboration between researchers at UC Berkeley, LBL, USC/ISI, and Xerox".