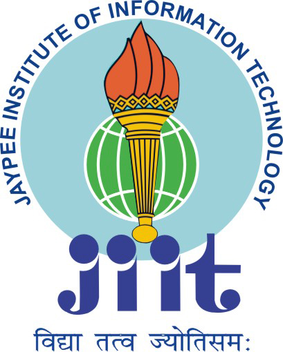
**NETWORK ON CHIPS**

Enrol. No. (s) - 9913103457, 9913103479, 9913103633

Name of Student (s) - Arnav Gupta, Abhratanu Paul, Mayank Bansal

Name of supervisor - Prof. Bansidhar Joshi



May – 2017

Submitted in partial fulfillment of the Degree of

Bachelor of Technology

in

Computer Science Engineering

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING & INFORMATION TECHNOLOGY

JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY, NOIDA

**(I)**

**TABLE OF CONTENTS**

**Chapter No. Topics Page No.**

Student Declaration II

Certificate from the Supervisor III

Acknowledgement IV

Summary V

List of Figures VI

List of Tables VII

List of Symbols and Acronyms VIII

**Chapter-1** **Introduction** Page No to Page No

1.1 General Introduction

1.2 Problem Statement

1.3 Empirical Study

1.4 Approach to problem in terms of technology /platform

1.5 Support for Novelty/ significance of problem

1.6 Other existing approaches/ solution to the problem framed

**Chapter-2** **Literature Survey** Page No to Page No

2.1 Summary of papers studied

2.2 Integrated summary of the literature studied

**Chapter 3: Analysis, Design and Modeling**  Page No to Page No

3.1 Overall description of the project

3.5 Design Diagrams

**Chapter-4 Implementation details and issues**

4.1 Implementation details and issues

4.1.1 Implementation Issues

4.1.2 Algorithms

4.2 Risk Analysis and Mitigation

**Chapter-5 Testing (Focus on Quality of Robustness and Testing)**

5.1 Testing Plan

5.2 Component decomposition and type of testing required

5.3 List all test cases in prescribed format

5.4 Error and Exception Handling

5.5 Limitations of the solution

**Chapter-6 Findings & Conclusion** Page No to Page No

6.1 Findings

6.2 Conclusion

6.3 Future Work

References IEEE Format (Listed alphabetically) Page No to Page No

**(II)**

**DECLARATION**

We hereby declare that this submission is our own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgment has been made in the text.

Place: JIIT-128, NOIDA Signature:

Date: 12 May, 2017 Name: Arnav Gupta, Abhratanu Paul, Mayank Bansal

Enrollment No: 9913103457, 9913103479, 9913103633

**(III)**

**CERTIFICATE**

This is to certify that the work titled “**NETWORK ON CHIPS**” submitted by “**Arnav Gupta, Abhratanu Paul, Mayank Bansal**” in partial fulfillment for the award of degree of Bachelor of Technology of Jaypee Institute of Information Technology, Noida has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

Signature of Supervisor ……………………..

Name of Supervisor ……………………..

Designation ……………………..

Date 12 May, 2017

**(IV)**

**ACKNOWLEDGEMENT**

I would like to place on record my deep sense of gratitude to Prof. Bansidhar Joshi, faculty, Jaypee Institute of Information Technology, Noida for his generous guidance, help and useful suggestions. His continuous encouragement and supervision throughout the course of present work is the main reason behind the success of the project.

Signature of the Student ……………………..

Name of Student ……………………..

Enrollment Number ……………………..

Date ……………………..

**(V)**

**SUMMARY**

With the number of cores of chip multiprocessor (CMP’s) rapidly growing as technology scales down, connecting the different component of a CMP in a scalable and efficient way becomes increasingly challenging. In this project we explore the architectural implications of Interconnection Network designs for CMPs. We evaluate and compare different topologies using Network on Chip using different simulators.

The primary focus of this report is to give a brief background on what is currently happening in the field of Network on Chip multiprocessors and study various simulators and implement new topologies. The second half of our project is to integrate gem5 simulator with Topaz simulator and implement our own new network Topologies and to compare various topologies on the basis of various parameters like Throughput, and power.

**(VI)**

**LIST OF FIGURES**

1. Figure 1 Buffer Size v/s End-to-End Delay for Static Routing
2. Figure 2 Buffer Size v/s Drop Rate for Static Routing
3. Figure 3 Buffer Size v/s Buffer Utilization for Static Routing
4. Figure 4 Buffer Size v/s End-to-End Delay for XY Routing
5. Figure 5 Buffer Size v/s Drop Rate for XY Routing
6. Figure 6 Buffer Size v/s Buffer Utilization for XY Routing
7. Figure 7 Buffer Size v/s End-to-End Delay for Distance Vector Routing
8. Figure 8 Buffer Size v/s Drop Rate for Distance Vector Routing
9. Figure 9 Buffer Size v/s Buffer Utilization for Distance Vector Routing
10. Figure 10 Buffer Size v/s End-to-End Delay for ACO Routing
11. Figure 11 Buffer Size v/s Drop Rate for ACO Routing
12. Figure 12 Buffer Size v/s Buffer Utilization for ACO Routing

**(VII)**

**LIST OF TABLES**

1. Table 1 Buffer Size v/s End-to-End Delay for Static Routing
2. Table 2 Buffer Size v/s Drop Rate for Static Routing
3. Table 3 Buffer Size v/s Buffer Utilization for Static Routing
4. Table 4 Buffer Size v/s End-to-End Delay for XY Routing
5. Table 5 Buffer Size v/s Drop Rate for XY Routing
6. Table 6 Buffer Size v/s Buffer Utilization for XY Routing
7. Table 7 Buffer Size v/s End-to-End Delay for Distance Vector Routing
8. Table 8 Buffer Size v/s Drop Rate for Distance Vector Routing
9. Table 9 Buffer Size v/s Buffer Utilization for Distance Vector Routing
10. Table 10 Buffer Size v/s End-to-End Delay for ACO Routing
11. Table 11 Buffer Size v/s Drop Rate for ACO Routing
12. Table 12 Buffer Size v/s Buffer Utilization for ACO Routing

**(VIII)**

**LIST OF SYMBOLS & ACRONYMS**

1. ACO - Ant Colony Optimization.

2. NoC - Network on Chip.

3. SoC- System on Chip.

4. I/O- Input Output.

5. CPU – Central Processing Unit.

6. HDD- Hard Disk Drive.

**Chapter-1 Introduction**

* 1. **General Introduction**

The demand for computer and system with high intensive application is growing rapidly with the advancement of technology and fast pace of work. To achieve the computational results within the minimum amount of time and with correctness of results requires a powerful and highly efficient system which can calculate such results within the expected time frame. When talking about a system the only expectation from it are low power consumption and high performance. A system comprises of hardware and software both to obtain these results. There are number of software present which provide accurate and precise results according to the need of the user. But when a hardware is considered the most basic component comes into use for computations are chips. Computing resources like CPU and IP’s are used to build the system in System-on-chip (SoC). The interconnection of these components between each other emerges as a major issue. So, when the number of requests increases a new interconnection approach to counter this limitation and overcome large wiring delays is to adopt network like interconnections known as Network-on-Chip (NoC) architecture. A NoC is constructed from multiple point-to-point data links interconnected by switches, such that messages can be relayed from any source module to any destination module over several links, by making routing decisions at the switches.

* 1. **Problem Statement**

Here in this research project work the emphasis lies on determining such Routing algorithm which provides the best trade-off result in finding efficient path to our data packets transmitted from source node to destination node. Basically the Routing algorithm must be efficient in terms that it should complement the topology used for a specified network. The topology and the routing algorithm must work in such a way that the delivery of the message can be ensured within the expected time frame without the loss of any information that is passed through this network.

* 1. **Empirical Study**

To begin the research there is field study that needs to be done to get the implementation of different topologies and routing algorithm for the dedicated Network-on-Chip architecture.

* The basic difference between a System-on-Chip and Network-on-Chip architectures. Why need of a new on-chip system? What are the limitation of using a System-on-Chip?
* What are topologies that are used for a SoC and a NoC architecture?
* Routing algorithms used in different scenarios to provide the best route for data packets to be transmitted from source to sink.
* Various tools are required to compute our result and offer us a solution with desired goals
* Study of different Simulators like NOXIM, TOPAZ, GEM5, NS2 etc.

**Noxim**

Noxim is a Network-on-Chip Simulator developed at the University of Catania (Italy). The Noxim simulator is developed using SystemC, a system description language based on C++. Noxim has a command line interface for defining several parameters of a NoC and it can be downloaded from SourceForge under GPL license terms.

* It is developed in SystemC.
* It has a command line interface for defining several parameters of a NoC.
* The user can customize the network size, buffer size, packet size distribution, routing algorithms, traffic time distribution.
* It allows NoC evaluation in terms of throughput, delay, and power consumption.
* The information is delivered to the user both in terms of average and per-communication results.
* The user is allowed to collect different evaluation metrics including the total number of received packets/flits, global average throughput, max/min global delay, total energy consumption, per communication delay /throughput /energy etc.

**Topaz**

* Topaz is a general-purpose interconnection network simulator that allows the modelling a wide variety of message routers with different tradeoffs between speed and precision.
* Topaz comes from SICOSYS simulator, which was originally conceived to obtain results.
* Implementation in C++ language.
* The simulator has support for parallel execution.
* Used in UNIX platform with a C++ standard compiler.

For the models provided, approximately 110 classes, distributed in about 50,000 lines of code have been necessary. The simulator has support for parallel execution using standard POSIX threads. The portability is high: can be used in **any UNIX** platform with a C++ standard compiler.

* 1. **Problem Approach**

**1.4.1 Routing**

Routing on NoC is quite similar to routing on any network. A routing algorithm determines how the data is routed from sender to receiver. Routing algorithms are divided into two groups, oblivious and adaptive algorithms. Oblivious algorithms are also divided into two subgroups: deterministic and stochastic algorithms. Oblivious algorithms route packets without any information about traffic amounts and conditions of the network, deterministic algorithms route packets always along a same route and stochastic routing is based on randomness.

**1.4.2** **Analyzing and working with existing Gem5 routing code**

We analyzed the routing related code available in Gem5 and found a table based routing and a XY routing snippet available. User can select at simulation invocation time which routing to use. We ran several simulations to get control results and get familiar with gem5 routing library. Next we analyzed the code in order to discover ways for making changes to existing code hence add a routing scheme not available in Gem5.

**1.4.3 Using Djikstra Algorithm**

We were successful in integrating Djikstra algorithm based routing with gem5 code. On running simulations and comparing results we discovered changes in key performance parameters – latency, throughput, energy, delay, drop rate and buffer utilization.

**Use in NoCs**

•We use Dijkstra’s algorithm to find the shortest path between the source and destination in topology graph in NoC.

•It is used for minimizing the traffic congestion rate.

**1.4.4 Using Ant-Colony Optimization Algorithm**

We were successful in implementing ant-colony algorithm based routing in ns2. On running simulations and comparing results we discovered changes in key performance parameters – delay, drop rate and buffer utilization.

**Use in NoCs**

•ACO is used in NoCs to minimize the saturation throughput and increase the area efficiency.

•It is used in congestion control and prevent deadlock.

**Chapter 2: Literature Survey**

**2.1 Literature Survey**

**2.1.1 Paper 1**

Title of paper – An Analysis of On-Chip Interconnection Networks for Large - Scale Chip Multiprocessors

Authors – Daniel Sanchez, George Michelogiannakis and Chritopher Kozyrakis, Stanford University

Year of publication –2015

Summary

This paper tells us interconnect is a major component in memory hierarchy and overall performance. Latency, not throughput, is the main interconnect performance constraint for these systems. The flattened butterfly outperforms the conventional mesh and fat tree topologies, mainly due to its reduced network latency. In terms of cost, all topologies have moderate area, and power requirements for the sizes we explore, but they have significant differences in scalability. The author has then discussed about the future research on large-scale CMPs should carefully consider the on-chip interconnect along the other components of the memory hierarchy.

**2.1.2 Paper 2**

Title of paper – Noxim: An Open, Extensible and Cycle-accurate Network On Simulator

Authors – Vincenzo Catania, Andrea Mineo, Salvatore Monteleone, Maurizi

University of Catania, and Kore University, Italy

Year of publication – 2014

Summary

This research paper explored the design space of a Network on Chip by assessing its power and performance requires the availability of fast and accurate simulation tools. In this paper we studied about Noxim, an open source, cycle accurate platform for the simulation of both conventional wire based NoCs and emerging WiNoC architectures.

**2.1.3 Paper 3**

Title of paper – Analysis of Network-on-Chip topologies for cost- efficient chip multiprocessors

Authors – Marta Ortin-Obon, Dario Suarez- Gracia, Maria villarroya-Gaud, University of Adelaid, Australia

Year of publication – 2015

Summary

This paper tells us that interconnection network and the cache hierarchy simultaneously helps identify improvement opportunities in the design of CMPs.

Both elements have significant influence on system performance, area, and power consumption.

**2.1.4 Paper 4**

Title of paper –Outstanding Research Problems in NoC Design: Circuit-, Microarchitecture-, and System-Level Perspectives

Authors – Radu Marculescu, Umit Y. Ogras, Li-Shiuan Peh, Natalie Enright Jerger, Yatin Hoskote

Year of publication – 2015

Summary

This paper discusses about a research area in the field of interconnection chip networks i.e. network on chip (NOC’s) which have a huge advantage over system on chip (SOC’s).This paper broadly describes 13 parameters related to infrastructure, optimization, mapping and communication analysis of NOC’s and further gives an elaborate description about the motivation, problem formulation, proposed approaches and open issues related to each parameter. Finally it tells about the evaluation and verification process after all these 13 problems are solved. SOC’s are considered not reliable due to bus interconnection, area and performance issues and the limitation is overcome by NOC’s as it enables communication between various components( example -processors, embedded memory, I/O cores)using a reliable topology.

**2.1.5 Paper 5**

Title of paper – Application-Aware Deadlock-Free Oblivious Routing

Authors – Michel Kinsy, Myong Hyon Cho, Tina Wen, Edward Suh†, Marten van Dijk, Srinivas Devadas

Year of publication – 2015

Summary

This paper discusses about the research area in the field of “Application aware deadlock free oblivious routing”. It basically tells about how oblivious routing frameworks used for simple and fast router designs that face many issues related to traffic patterns, bandwidth demands and deadlocks are optimized to meet the application communication characteristics. Therefore, a variety of algorithms are proposed with a focus on generic network architecture for oblivious routing.

**Chapter 3:Analysis, Design and Modelling**

**3.1 Overall Description**

The project basically deals in computing the most efficient routing algorithm or in more precise form Adaptive routing algorithm which are based on nature inspired phenomena like swarm intelligence.

The most popular Adaptive routing algorithm proposed to the best suitable route for packet delivery is Ant Colony Optimization. Our research work relies on comparing different routing algorithm flow like distance vector, link state and session based algorithm for different topologies as Mesh, KingMesh, and torus networks.

To evaluate the comparison among the variety of routing algorithms we calculate performance based parameters like latency, throughput, power consumption, communication load, and buffer utilization and obtain the result for most effective and highly performing routing algorithm.

**3.2 Requirement Specifications**

**Software Requirements**

Ubuntu LTS 14.04

Virtual Box

Web Browser

Simulators: Noxim, Topaz, Gem5

**Hardware Requirements**

A computer

Processor: Pentium 2.0 GHz or higher.

At least 20GB free HDD space for proper functioning

Ram 2GB or more.

Internet provider

Ethernet cable

**3.3 Functional Requirement**

* Provides simulation statistics of different topologies
* Compares different topologies and find the best according to different parameters (sim\_ticks, power, energy, latency).
* Networking testing and regression testing of different topologies.

**3.4 Non-Functional Requirement**

* Use of classic memory system instead of ruby.
* Use of garnet network instead of simple

**Chapter 4: Implementation details and Issues**

**4.1 Implementation details and Issues**

Start with creating directory of routing protocol. Go to the *“$NS\_ROOT/ ns-2.34/”*. Create directory named as **wfrp**, we call it WSN Flooding Based Routing Protocol in which sink nodes periodically send a beacon message and other nodes construct route towards the sink nodes. Then nodes report to sink node every certain period using UDP protocol. Direct Diffusion may be an example of such protocol, but what we are writing is simpler and has more functionalities.

|  |  |
| --- | --- |
| 1 | mkdir wfrp |

In the directory we create three files: [wrfp.cc](http://elmurod.net/sfiles/wfrp/wfrp.cc), wrfp.h, wrfp\_packet.h. Download and put these files in **wfrp** directory. I will not explain the code here, and if you don’t understand just leave comment I will try to answer.

Now, we are going to modify following files. Therefore it is better you backup these files before you start adding new protocol, so that you can easily go back.

* $NS\_ROOT/Makefile
* $NS\_ROOT/queue/priqueue.cc
* $NS\_ROOT/common/packet.h
* $NS\_ROOT/trace/cmu-trace.h
* $NS\_ROOT/trace/cmu-trace.cc
* $NS\_ROOT/tcl/lib/ns-packet.tcl
* $NS\_ROOT/tcl/lib/ns-lib.tcl
* $NS\_ROOT/tcl/lib/ns-agent.tcl
* $NS\_ROOT/tcl/lib/ns-mobilenode.tcl

Start with ~/ns-allinone-2.34/ns-2.34/**Makefile** just add following line at 269

|  |  |
| --- | --- |
| 1 | wfrp/wfrp.o |

Add following lines to ~/ns-allinone-2.34/ns-2.34/queue/**priqueue.cc** from line 93.

|  |  |
| --- | --- |
| 1 | // WFRP patch |
| 2 | case PT\_WFRP: |

To define new routing protocol packet type we have to modify ~/ns-allinone-2.34/ns-2.34/common/**packet.h** file. We change PT\_NTYPE to 63, and for our protocol PT\_WFRP = 62. If you have already installed another routing protocol. Just make sure PT\_NTYPE is last, and protocol number is ordered sequentially. From line 85 changes would be:

|  |  |
| --- | --- |
| 1 | // WFRP packet |
| 2 | static const packet\_t PT\_WFRP = 62; |

|  |  |
| --- | --- |
| 3 |  |
| 4 | // insert new packet types here |

|  |  |
| --- | --- |
| 5 | static packet\_t PT\_NTYPE = 63; // This MUST be the LAST one |

We make following code change at line 254 of ~/ns-allinone-2.34/ns-2.34/common/packet.h. The code is used that the packet is routing protocol packet and has high priority.

|  |  |
| --- | --- |
| 1 | type == PT\_AODV || |
| 2 | type == PT\_WFRP) |

And at line 390 of the same file

|  |  |
| --- | --- |
| 1 | // WFRP patch |
| 2 | name\_[PT\_WFRP] = "WFRP"; |

Now we will make NS2 trace our simulation and write it to \*something\*.tr, in order to do that we have to modify cmu-trace.h and cmu-trace.cc.

To add trace function we add following line to ~/ns-allinone-2.34/ns-2.34/trace/**cmu-trace.h** at line 163:

|  |  |
| --- | --- |
| 1 | void    format\_wfrp(Packet \*p, int offset); |

~/ns-allinone-2.34/ns-2.34/trace/**cmu-trace.cc** must be added following code at line 1071

|  |  |
| --- | --- |
| 1 | // WFRP patch |
| 2 | void |

|  |  |
| --- | --- |
| 3 | CMUTrace::format\_wfrp(Packet \*p, int offset) |
| 4 | { |

|  |  |
| --- | --- |
| 5 | struct hdr\_wfrp \*wh = HDR\_WFRP(p); |
| 6 | struct hdr\_wfrp\_beacon \*wb = HDR\_WFRP\_BEACON(p); |

|  |  |
| --- | --- |
| 7 | struct hdr\_wfrp\_error  \*we = HDR\_WFRP\_ERROR(p); |
| 8 |  |

|  |  |  |
| --- | --- | --- |
| 9 | switch(wh->pkt\_type) { | |
| 10 | | case WFRP\_BEACON: |

|  |  |
| --- | --- |
| 11 |  |
| 12 | if (pt\_->tagged()) { |

|  |  |
| --- | --- |
| 13 | sprintf(pt\_->buffer() + offset, |
| 14 | "-wfrp:t %x -wfrp:h %d -wfrp:b %d -wfrp:s %d " |

|  |  |
| --- | --- |
| 15 | "-wfrp:px %d -wfrp:py %d -wfrp:ts %f " |
| 16 | "-wfrp:c BEACON ", |

|  |  |
| --- | --- |
| 17 | wb->pkt\_type, |
| 18 | wb->beacon\_hops, |

|  |  |
| --- | --- |
| 19 | wb->beacon\_id, |
| 20 | wb->beacon\_src, |

|  |  |
| --- | --- |
| 21 | wb->beacon\_posx, |
| 22 | wb->beacon\_posy, |

|  |  |
| --- | --- |
| 23 | wb->timestamp); |
| 24 | } else if (newtrace\_) { |

|  |  |
| --- | --- |
| 25 |  |
| 26 | sprintf(pt\_->buffer() + offset, |

|  |  |
| --- | --- |
| 27 | "-P wfrp -Pt 0x%x -Ph %d -Pb %d -Ps %d -Ppx %d -Ppy %d -Pts %f -Pc BEACON ", |
| 28 | wb->pkt\_type, |

|  |  |
| --- | --- |
| 29 | wb->beacon\_hops, |
| 30 | wb->beacon\_id, |

|  |  |
| --- | --- |
| 31 | wb->beacon\_src, |
| 32 | wb->beacon\_posx, |

|  |  |
| --- | --- |
| 33 | wb->beacon\_posy, |
| 34 | wb->timestamp); |

|  |  |
| --- | --- |
| 35 |  |
| 36 | } else { |

|  |  |
| --- | --- |
| 37 |  |
| 38 | sprintf(pt\_->buffer() + offset, |

|  |  |
| --- | --- |
| 39 | "[0x%x %d %d [%d %d] [%d %f]] (BEACON)", |
| 40 | wb->pkt\_type, |

|  |  |
| --- | --- |
| 41 | wb->beacon\_hops, |
| 42 | wb->beacon\_id, |

|  |  |
| --- | --- |
| 43 | wb->beacon\_src, |
| 44 | wb->beacon\_posx, |

|  |  |
| --- | --- |
| 45 | wb->beacon\_posy, |
| 46 | wb->timestamp); |

|  |  |
| --- | --- |
| 47 | } |
| 48 | break; |

|  |  |
| --- | --- |
| 49 |  |
| 50 | case WFRP\_ERROR: |

|  |  |
| --- | --- |
| 51 | // TODO: need to add code |
| 52 | break; |

|  |  |
| --- | --- |
| 53 |  |
| 54 | default: |

|  |  |
| --- | --- |
| 55 | #ifdef WIN32 |
| 56 | fprintf(stderr, |

|  |  |
| --- | --- |
| 57 | "CMUTrace::format\_wfrp: invalid WFRP packet typen"); |
| 58 | #else |

|  |  |
| --- | --- |
| 59 | fprintf(stderr, |
| 60 | "%s: invalid WFRP packet typen", \_\_FUNCTION\_\_); |

|  |  |
| --- | --- |
| 61 | #endif |
| 62 | abort(); |

|  |  |
| --- | --- |
| 63 | } |
| 64 | } |

Now we will modify tcl files to create routing agent. First we define protocol name to use in tcl file. It would done by modifying ~/ns-allinone-2.34/ns-2.34/tcl/lib/**ns-packet.tcl** @ line 172

|  |  |
| --- | --- |
| 1 | # WFRP patch |
| 2 | WFRP |

Now we set routing agent by modifying ~/ns-allinone-2.34/ns-2.34/tcl/lib/**ns-lib.tcl** @ line 633

|  |  |
| --- | --- |
| 1 | WFRP { |
| 2 | set ragent [$self create-wfrp-agent $node] |

|  |  |
| --- | --- |
| 3 | } |

From line 860 of the same file following code should be added.

|  |  |
| --- | --- |
| 1 | Simulator instproc create-wfrp-agent { node } { |
| 2 | #  Create WFRP routing agent |

|  |  |
| --- | --- |
| 3 | set ragent [new Agent/WFRP [$node node-addr]] |
| 4 | $self at 0.0 "$ragent start" |

|  |  |
| --- | --- |
| 5 | $node set ragent\_ $ragent |
| 6 | return $ragent |

|  |  |
| --- | --- |
| 7 | } |

Now we will set port numbers of routing agent. sport is source port, dport is destination port. Modify ~/ns-allinone-2.34/ns-2.34/tcl/lib/**ns-agent.tcl** line 202

|  |  |
| --- | --- |
| 1 | Agent/WFRP instproc init args { |
| 2 | $self next $args |

|  |  |
| --- | --- |
| 3 | } |
|  |  |

|  |  |
| --- | --- |
| 5 | Agent/WFRP set sport\_   0 |
| 6 | Agent/WFRP set dport\_   0 |

~/ns-allinone-2.34/ns-2.34/tcl/lib/**ns-mobilenode.tcl** line 201

|  |  |
| --- | --- |
| 1 | # Special processing for WFRP |
| 2 | set wfrponly [string first "WFRP" [$agent info class]] |

|  |  |
| --- | --- |
| 3 | if {$wfrponly != -1 } { |
| 4 | $agent if-queue [$self set ifq\_(0)]   ;# ifq between LL and MAC |

|  |  |
| --- | --- |
| 5 | } |

We are done. got to ~/ns-allinone-2.34/ns-2.34/ directory and do

|  |  |
| --- | --- |
| 1 | make clean |
| 2 | make |

When the compile is finished, you can test using wfrp\_802\_15\_4.tcl file as:

|  |  |
| --- | --- |
| 1 | ns wfrp\_802\_15\_4.tcl |

In this test the NODE 0 is *sink node,* starts sending beacon 1 second after simulation i started, and NODE 10 is reporting node. It starts sending report over CBR/UDP at 5.0 seconds (after simulation is started). Report interval is 2 seconds.

**4.1.1 Implementation Issues**

The problems while implementing different routing algorithm that arise are the following:

* Lack of comprehensive implementation guides for NoC related topics.
* Machine limitations to run simulations for large topologies
* A gap between available NoC literature and feasibility of its implementation.

**4.1.2 Algorithms**

**XY Routing-** XY Routing is like dimension order routing which routes the data packet or message firstly in the X or horizontal direction to the correct column and the finally route it to the Y- or vertical direction to the destination or receiver node. For a network like mesh or torus it is well suited due to its high and effective performance than DOR. Addresses of the routers act as the XY coordinate for the routing purpose. But there is still problems with the traditional XY routing. The traffic does not extend regularly over the whole network because the algorithm causes the biggest load in the middle of the network. There is a need for algorithms which equalize the traffic load over the whole network.

**Dijkstra’s Algorithm-** Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks.

The algorithm exists in many variants; Dijkstra's original variant found the shortest path between two nodes, but a more common variant fixes a single node as the "source" node and finds shortest paths from the source to all other nodes in the graph, producing a shortest-path tree. For a given source node in the graph, the algorithm finds the shortest path between that node and every other. It can also be used for finding the shortest paths from a single node to a single destination node by stopping the algorithm once the shortest path to the destination node has been determined.

**Ant Colony Optimization**- In software engineering and operations explore, the Ant Colony Optimization algorithm (ACO) is a probabilistic procedure for tackling computational issues which can be lessened to discovering great ways through graphs and diagrams.

This calculation is an individual from the subterranean insect settlement calculations family, in swarm knowledge techniques, and it constitutes some meta-heuristic advancements. The first thought has since broadened to illuminate a more extensive class of numerical issues, and subsequently, a few issues have developed, drawing on different parts of the conduct of ants. From a more extensive viewpoint, ACO plays out a model-based pursuit and impart a few likenesses to Estimation of Distribution Algorithms.

**4.2 Risk Analysis**

The risks related to this project are following:

1. The dataset of mapped emotions has not been provided anywhere.
2. Particular research has not been done on the topic of emotion deduction and hence requires a thorough research.
3. There are a lot of variations in the emotions at a particular instant of time.

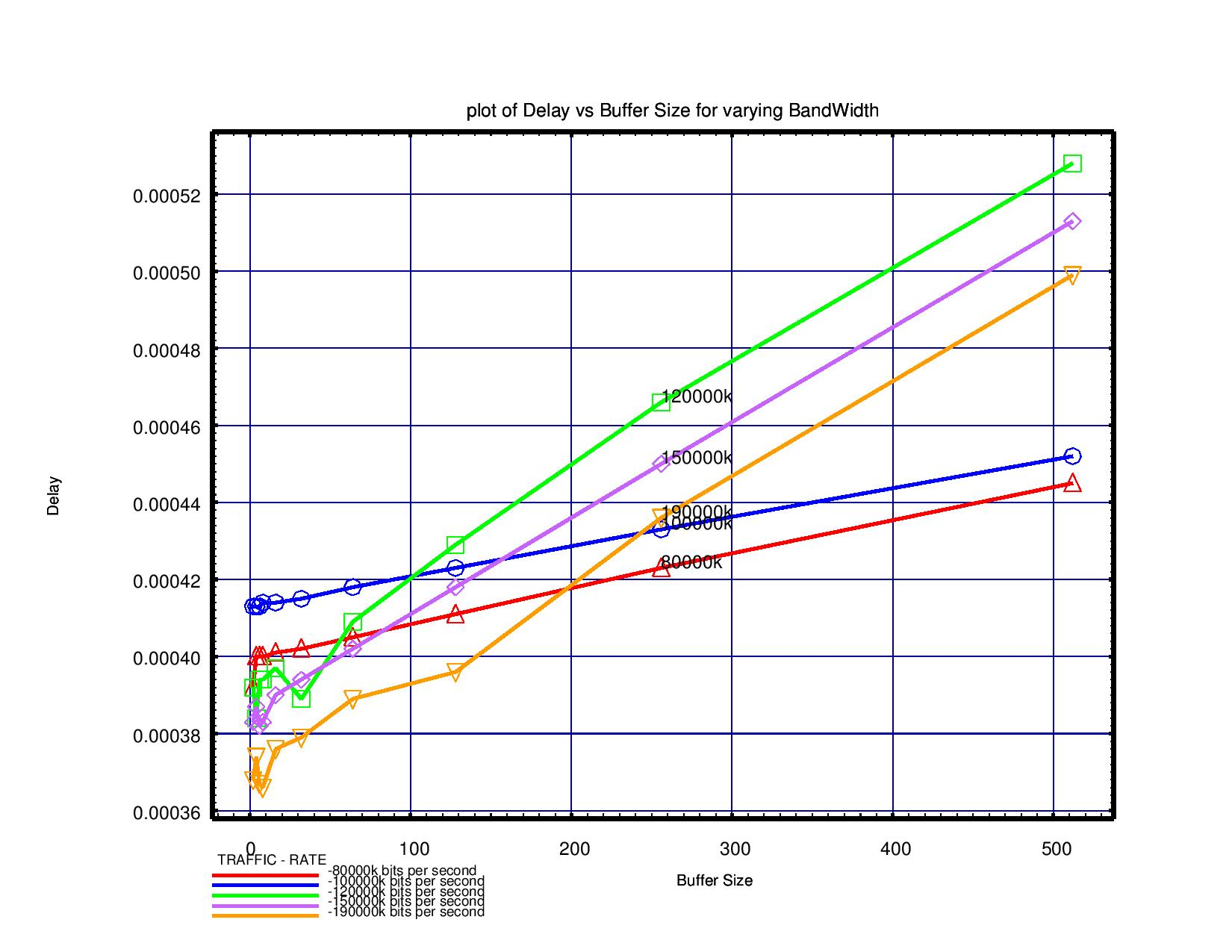
**Chapter 5: Testing**

**5.1 Test results**

**5.1.1 Static Routing**

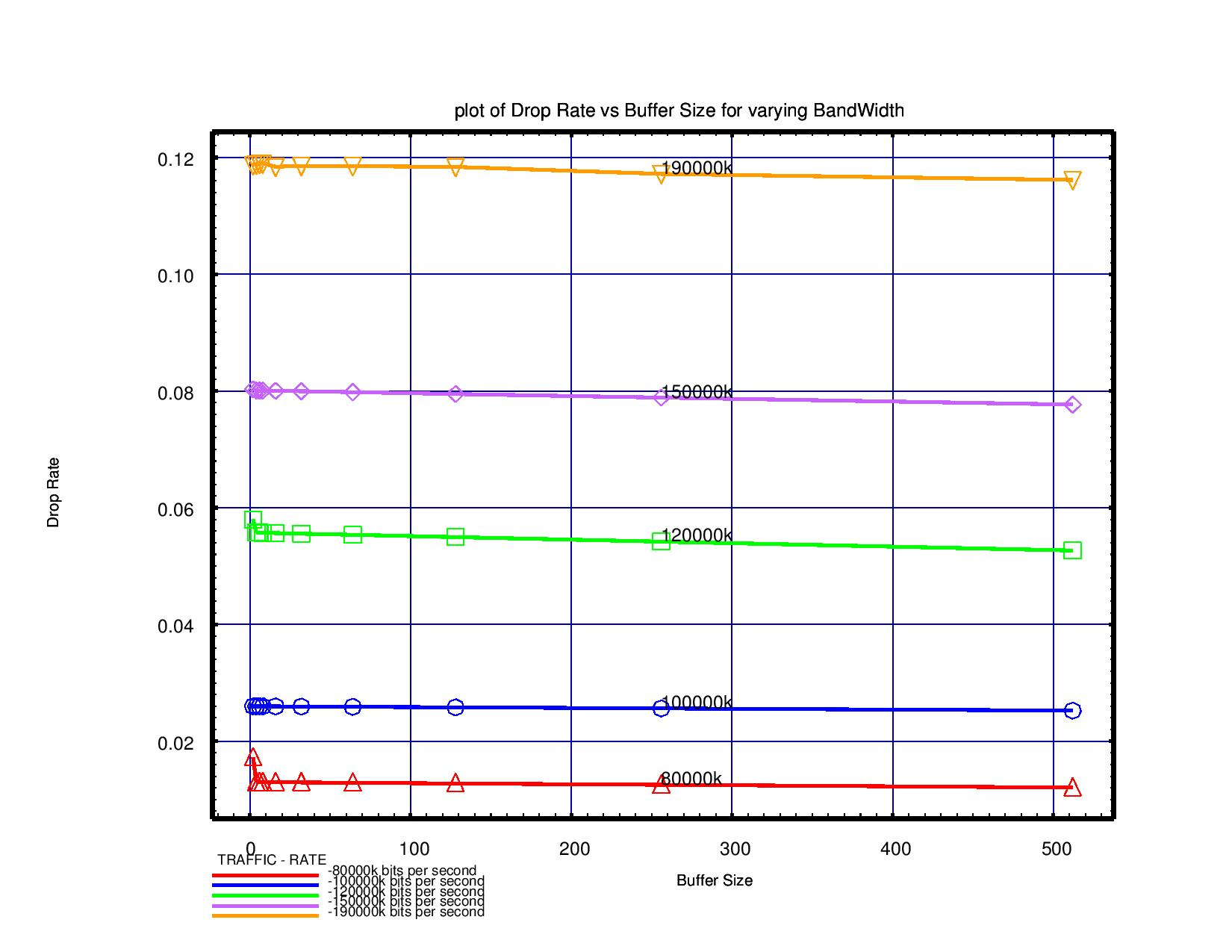
Buffer size v/s End-to-End Delay

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Buffer Size | Delay(micro secs) for traffic rates | | | | |
| 80000k | 100000k | 120000k | 150000k | 190000k |
| 2 | 0.000392 | 0.000413 | 0.000392 | 0.000383 | 0.000368 |
| 4 | 0.0004 | 0.000413 | 0.000384 | 0.000387 | 0.000374 |
| 6 | 0.0004 | 0.000413 | 0.000394 | 0.000382 | 0.000367 |
| 8 | 0.0004 | 0.000414 | 0.000394 | 0.000383 | 0.000366 |
| 16 | 0.000401 | 0.000414 | 0.000397 | 0.00039 | 0.000376 |
| 32 | 0.000402 | 0.000415 | 0.000389 | 0.000394 | 0.000379 |
| 64 | 0.000405 | 0.000418 | 0.000409 | 0.000402 | 0.000389 |
| 128 | 0.000411 | 0.000423 | 0.000429 | 0.000418 | 0.000396 |
| 256 | 0.000423 | 0.000433 | 0.000466 | 0.00045 | 0.000436 |
| 512 | 0.000445 | 0.000452 | 0.000528 | 0.000513 | 0.000499 |



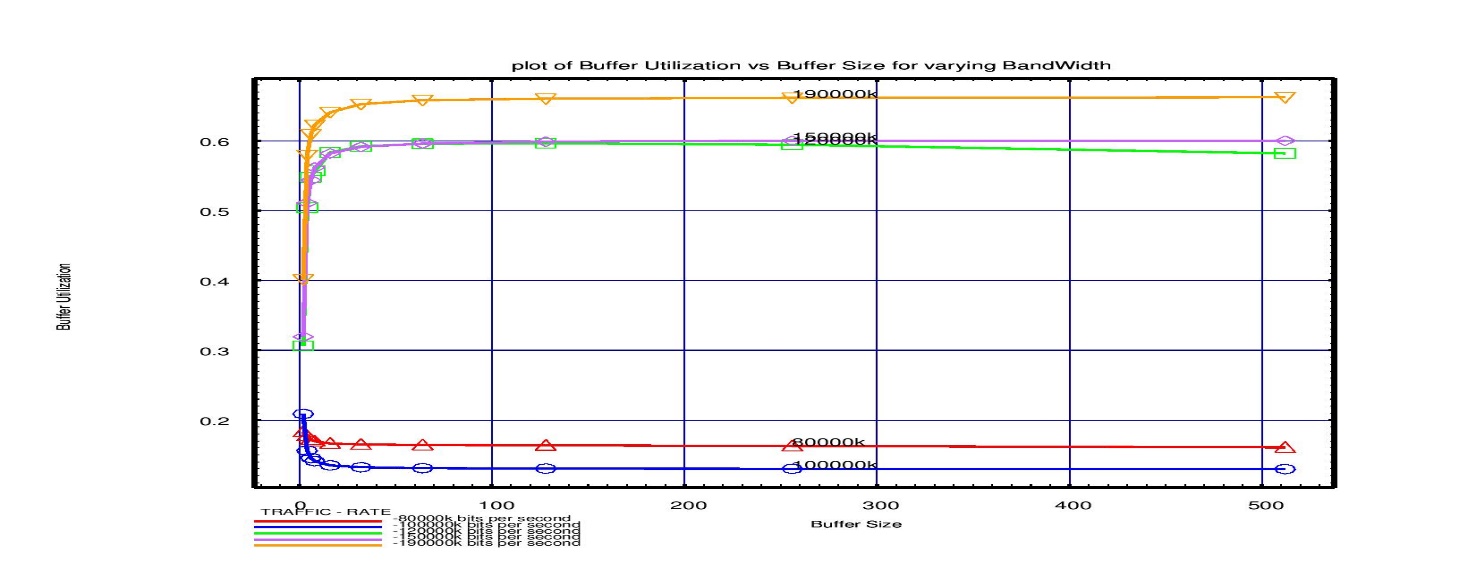
Buffer size v/s Drop Rate

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Buffer Size | Drop Rate for traffic rates | | | | |
| 80000k | 100000k | 120000k | 150000k | 190000k |
| 2 | 0.017217 | 0.025981 | 0.057995 | 0.080299 | 0.118701 |
| 4 | 0.012987 | 0.025978 | 0.055731 | 0.080087 | 0.118836 |
| 6 | 0.012983 | 0.025975 | 0.055718 | 0.080078 | 0.118972 |
| 8 | 0.012979 | 0.025972 | 0.055706 | 0.080068 | 0.119006 |
| 16 | 0.012965 | 0.025961 | 0.055657 | 0.08003 | 0.118413 |
| 32 | 0.012936 | 0.025937 | 0.05556 | 0.079953 | 0.118592 |
| 64 | 0.012877 | 0.025891 | 0.055366 | 0.0798 | 0.11853 |
| 128 | 0.012761 | 0.025798 | 0.054977 | 0.079493 | 0.118392 |
| 256 | 0.012528 | 0.025611 | 0.0542 | 0.078882 | 0.1172 |
| 512 | 0.012061 | 0.025238 | 0.052677 | 0.077661 | 0.116158 |



Buffer size v/s Buffer Utilization

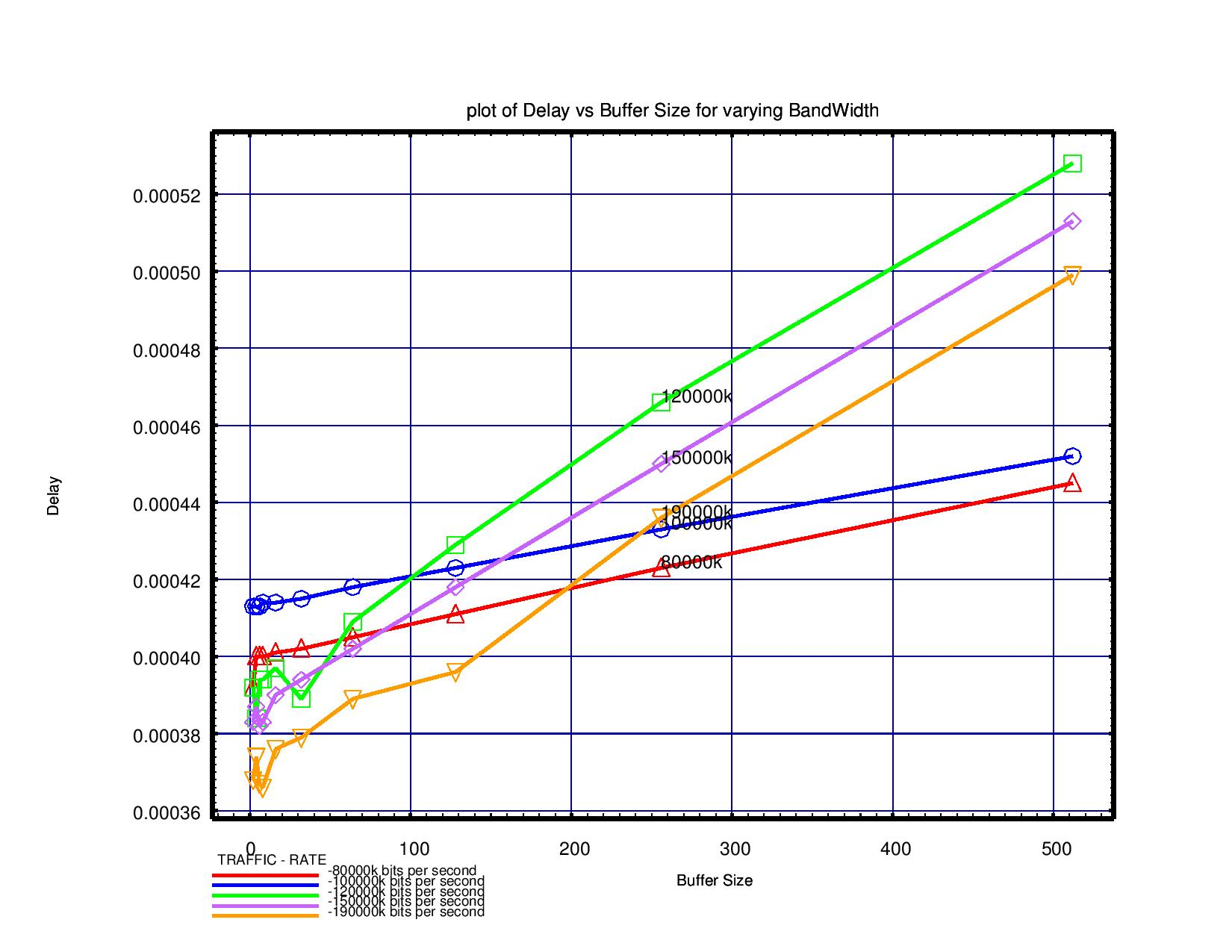
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Buffer Size | Buffer Utilization for traffic rates | | | | |
| 80000k | 100000k | 120000k | 150000k | 190000k |
| 2 | 0.183004 | 0.208721 | 0.306352 | 0.319264 | 0.401556 |
| 4 | 0.177291 | 0.156121 | 0.504522 | 0.511109 | 0.579474 |
| 6 | 0.171883 | 0.145601 | 0.547694 | 0.543757 | 0.609076 |
| 8 | 0.169562 | 0.141092 | 0.55691 | 0.561727 | 0.622811 |
| 16 | 0.166453 | 0.135081 | 0.583368 | 0.58205 | 0.640918 |
| 32 | 0.165018 | 0.132366 | 0.592028 | 0.591376 | 0.652624 |
| 64 | 0.164272 | 0.131073 | 0.595724 | 0.596108 | 0.658214 |
| 128 | 0.163703 | 0.130442 | 0.596527 | 0.598832 | 0.660367 |
| 256 | 0.162763 | 0.130107 | 0.59445 | 0.600127 | 0.661601 |
| 512 | 0.160779 | 0.129781 | 0.581959 | 0.600172 | 0.66241 |



**5.1.2 Session Based Routing**

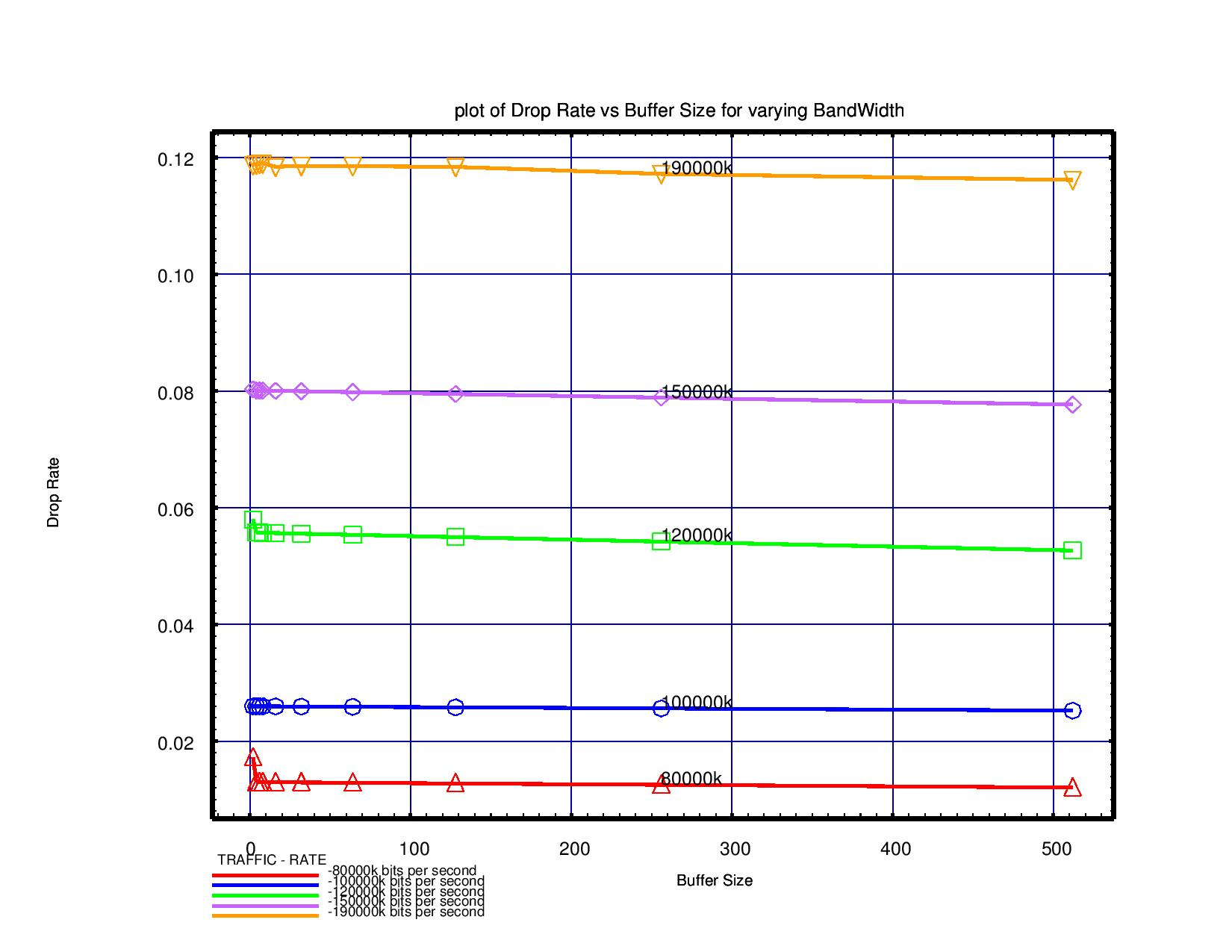
Buffer Size v/s End-to-End Delay

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Buffer Size | Delay(micro secs) for traffic rates | | | | |
| 80000k | 100000k | 120000k | 150000k | 190000k |
| 2 | 0.000392 | 0.000413 | 0.000392 | 0.000383 | 0.000368 |
| 4 | 0.0004 | 0.000413 | 0.000384 | 0.000387 | 0.000374 |
| 6 | 0.0004 | 0.000413 | 0.000394 | 0.000382 | 0.000367 |
| 8 | 0.0004 | 0.000414 | 0.000394 | 0.000383 | 0.000366 |
| 16 | 0.000401 | 0.000414 | 0.000397 | 0.00039 | 0.000376 |
| 32 | 0.000402 | 0.000415 | 0.000389 | 0.000394 | 0.000379 |
| 64 | 0.000405 | 0.000418 | 0.000409 | 0.000402 | 0.000389 |
| 128 | 0.000411 | 0.000423 | 0.000429 | 0.000418 | 0.000396 |
| 256 | 0.000423 | 0.000433 | 0.000466 | 0.00045 | 0.000436 |
| 512 | 0.000445 | 0.000452 | 0.000528 | 0.000513 | 0.000499 |



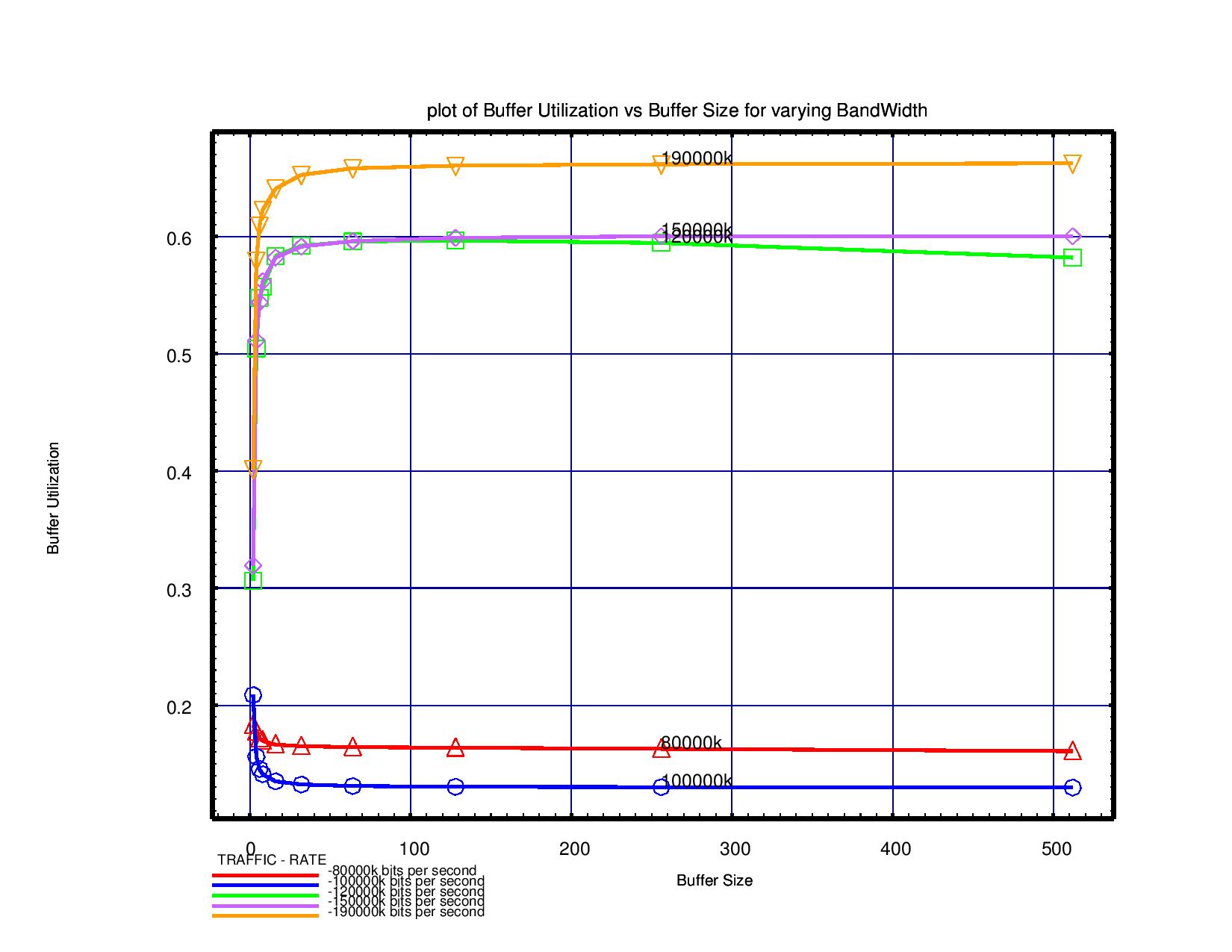
Buffer Size v/s Drop Rate

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Buffer Size | Drop Rate for traffic rates | | | | |
| 80000k | 100000k | 120000k | 150000k | 190000k |
| 2 | 0.017217 | 0.025981 | 0.057995 | 0.080299 | 0.118701 |
| 4 | 0.012987 | 0.025978 | 0.055731 | 0.080087 | 0.118836 |
| 6 | 0.012983 | 0.025975 | 0.055718 | 0.080078 | 0.118972 |
| 8 | 0.012979 | 0.025972 | 0.055706 | 0.080068 | 0.119006 |
| 16 | 0.012965 | 0.025961 | 0.055657 | 0.08003 | 0.118413 |
| 32 | 0.012936 | 0.025937 | 0.05556 | 0.079953 | 0.118592 |
| 64 | 0.012877 | 0.025891 | 0.055366 | 0.0798 | 0.11853 |
| 128 | 0.012761 | 0.025798 | 0.054977 | 0.079493 | 0.118392 |
| 256 | 0.012528 | 0.025611 | 0.0542 | 0.078882 | 0.1172 |
| 512 | 0.012061 | 0.025238 | 0.052677 | 0.077661 | 0.116158 |



Buffer Size v/s Buffer Utilization

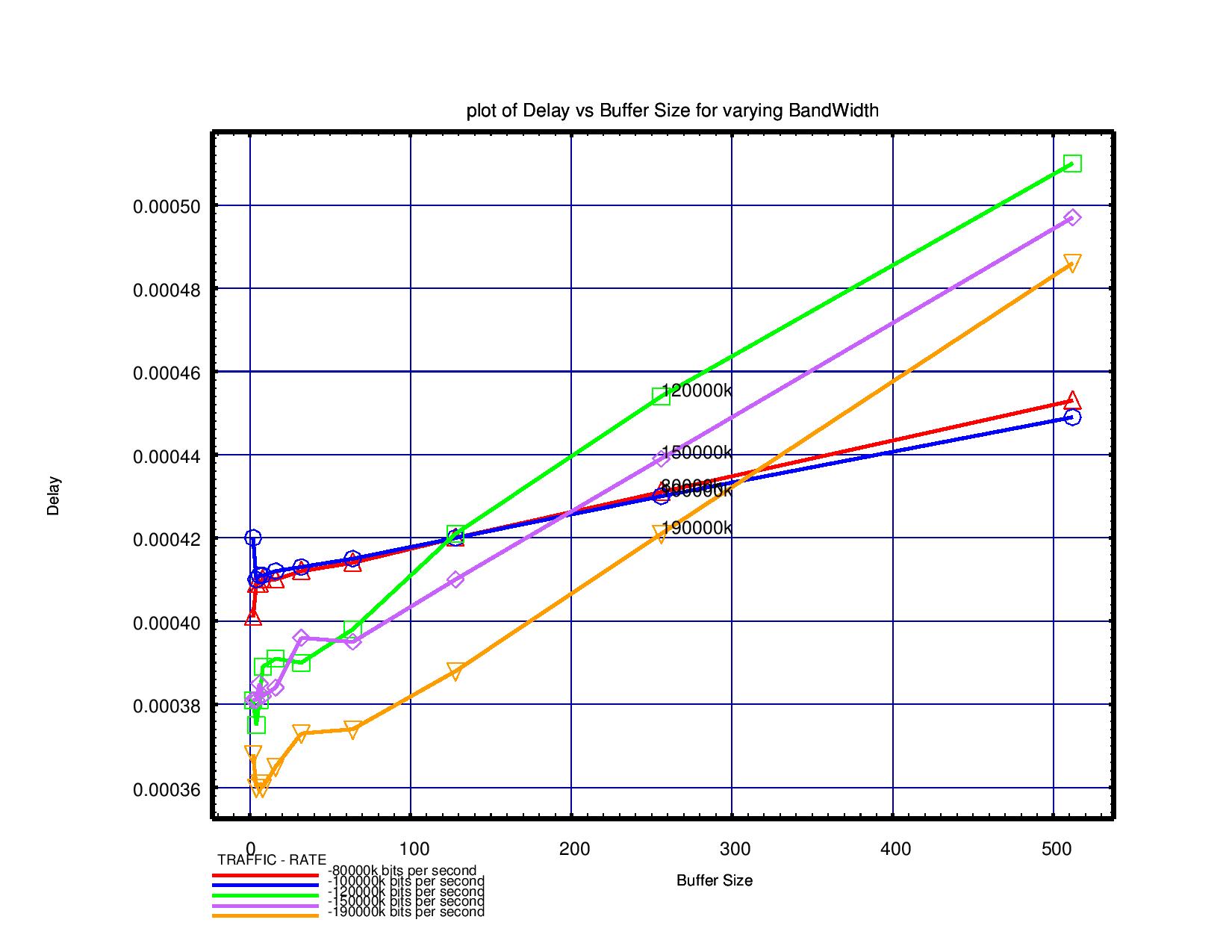
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Buffer Size | Buffer Utilization for traffic rates | | | | |
| 80000k | 100000k | 120000k | 150000k | 190000k |
| 2 | 0.183004 | 0.208721 | 0.306352 | 0.319264 | 0.401556 |
| 4 | 0.177291 | 0.156121 | 0.504522 | 0.511109 | 0.579474 |
| 6 | 0.171883 | 0.145601 | 0.547694 | 0.543757 | 0.609076 |
| 8 | 0.169562 | 0.141092 | 0.55691 | 0.561727 | 0.622811 |
| 16 | 0.166453 | 0.135081 | 0.583368 | 0.58205 | 0.640918 |
| 32 | 0.165018 | 0.132366 | 0.592028 | 0.591376 | 0.652624 |
| 64 | 0.164272 | 0.131073 | 0.595724 | 0.596108 | 0.658214 |
| 128 | 0.163703 | 0.130442 | 0.596527 | 0.598832 | 0.660367 |
| 256 | 0.162763 | 0.130107 | 0.59445 | 0.600127 | 0.661601 |
| 512 | 0.160779 | 0.129781 | 0.581959 | 0.600172 | 0.66241 |



**5.1.3 Distance Vector Routing**

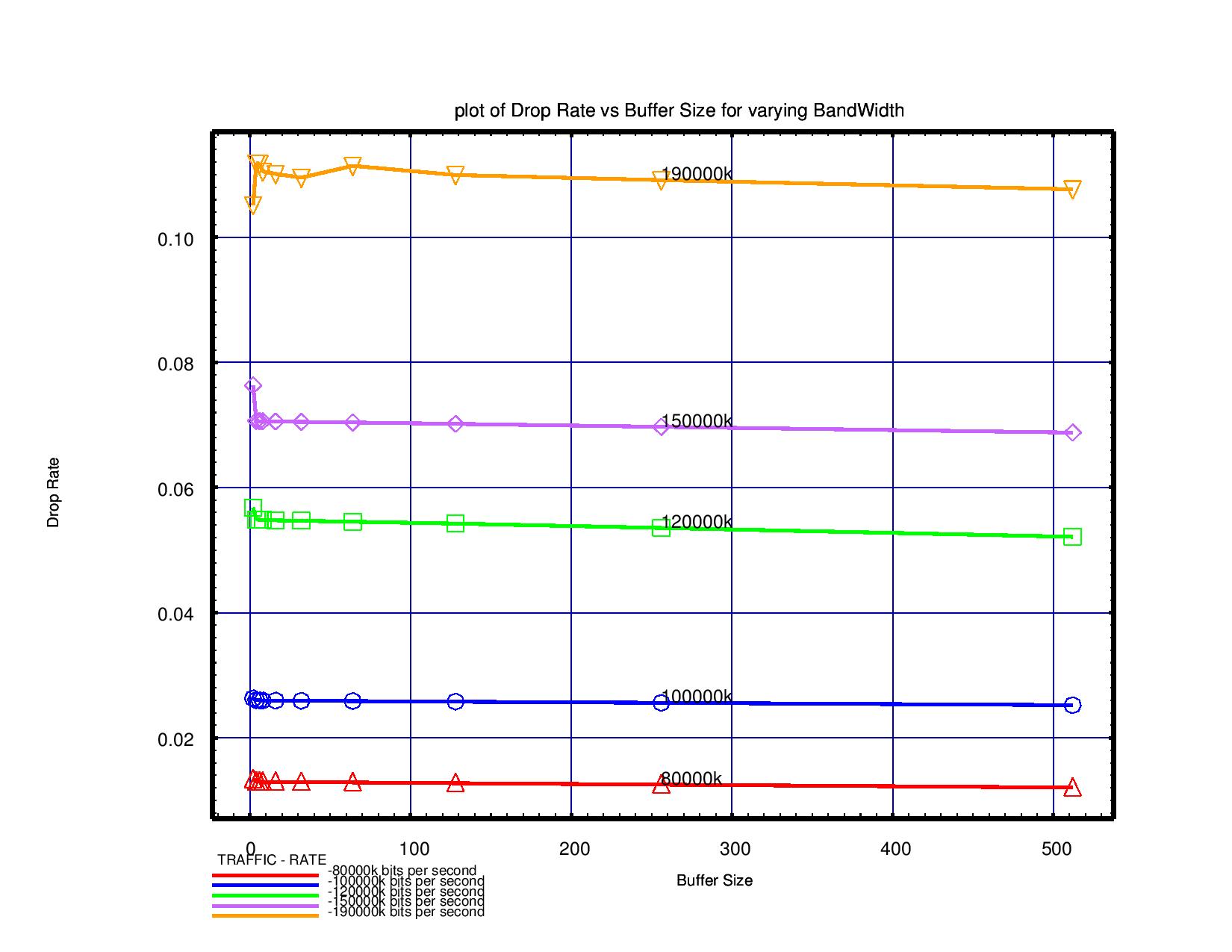
Buffer Size v/s End-to-End Delay

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Buffer Size | Delay(micro secs) for traffic rates | | | | |
| 80000k | 100000k | 120000k | 150000k | 190000k |
| 2 | 0.000401 | 0.00042 | 0.000381 | 0.000381 | 0.000368 |
| 4 | 0.000409 | 0.00041 | 0.000375 | 0.000381 | 0.00036 |
| 6 | 0.000409 | 0.000411 | 0.000381 | 0.000385 | 0.000361 |
| 8 | 0.00041 | 0.000411 | 0.000389 | 0.000382 | 0.00036 |
| 16 | 0.00041 | 0.000412 | 0.000391 | 0.000384 | 0.000365 |
| 32 | 0.000412 | 0.000413 | 0.00039 | 0.000396 | 0.000373 |
| 64 | 0.000414 | 0.000415 | 0.000398 | 0.000395 | 0.000374 |
| 128 | 0.00042 | 0.00042 | 0.000421 | 0.00041 | 0.000388 |
| 256 | 0.000431 | 0.00043 | 0.000454 | 0.000439 | 0.000421 |
| 512 | 0.000453 | 0.000449 | 0.00051 | 0.000497 | 0.000486 |



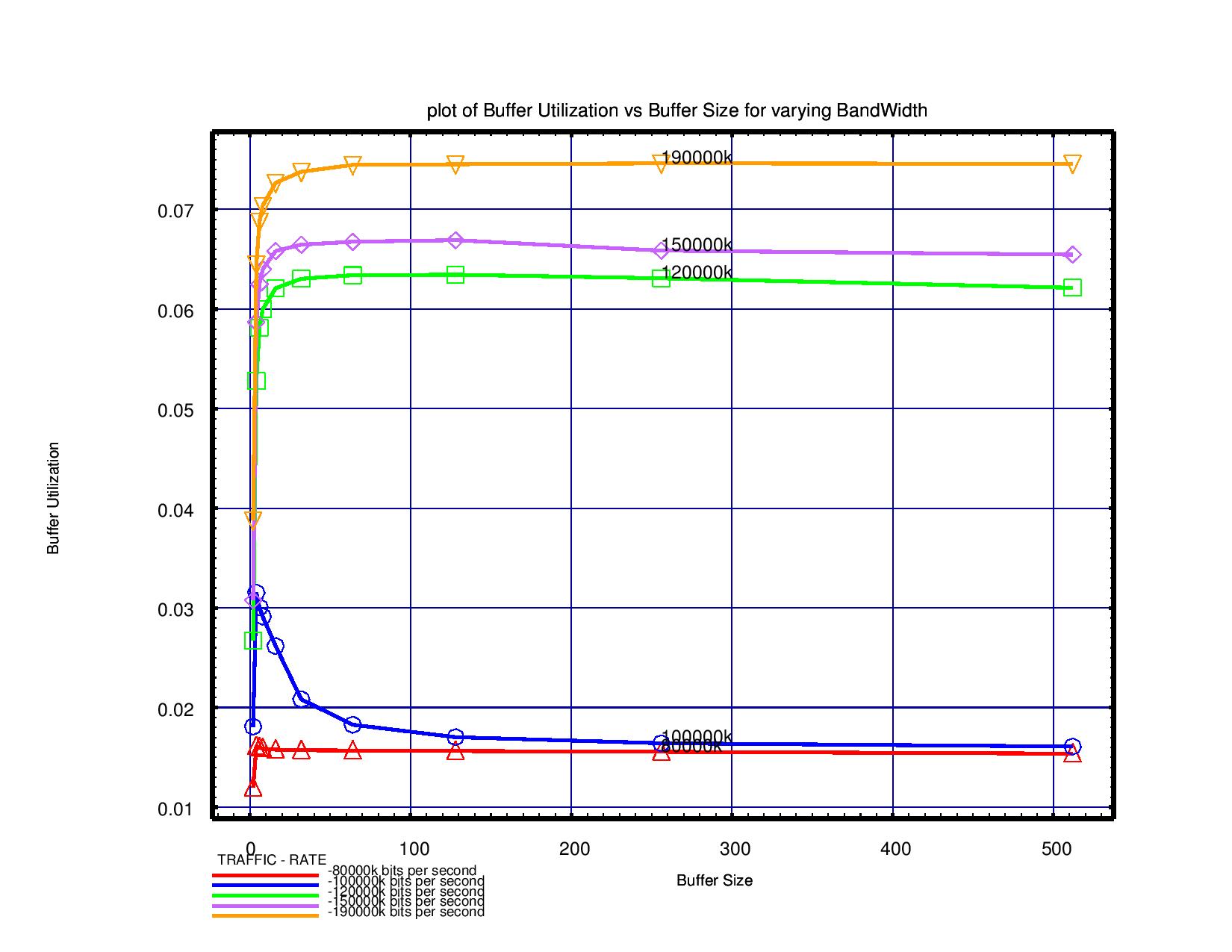
Buffer Size v/s Drop Rate

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Buffer Size | Drop Rate for traffic rates | | | | |
| 80000k | 100000k | 120000k | 150000k | 190000k |
| 2 | 0.013354 | 0.026297 | 0.05675 | 0.076328 | 0.105158 |
| 4 | 0.013016 | 0.02601 | 0.054881 | 0.070638 | 0.11182 |
| 6 | 0.01298 | 0.025982 | 0.054849 | 0.070608 | 0.111774 |
| 8 | 0.012967 | 0.025965 | 0.054825 | 0.070591 | 0.110525 |
| 16 | 0.01295 | 0.025944 | 0.054769 | 0.070555 | 0.110139 |
| 32 | 0.012921 | 0.025921 | 0.054693 | 0.070497 | 0.109537 |
| 64 | 0.012863 | 0.025874 | 0.054544 | 0.0704 | 0.111424 |
| 128 | 0.012746 | 0.025781 | 0.054239 | 0.070184 | 0.109958 |
| 256 | 0.012513 | 0.025594 | 0.053552 | 0.069708 | 0.109129 |
| 512 | 0.012048 | 0.025222 | 0.052125 | 0.068788 | 0.107677 |



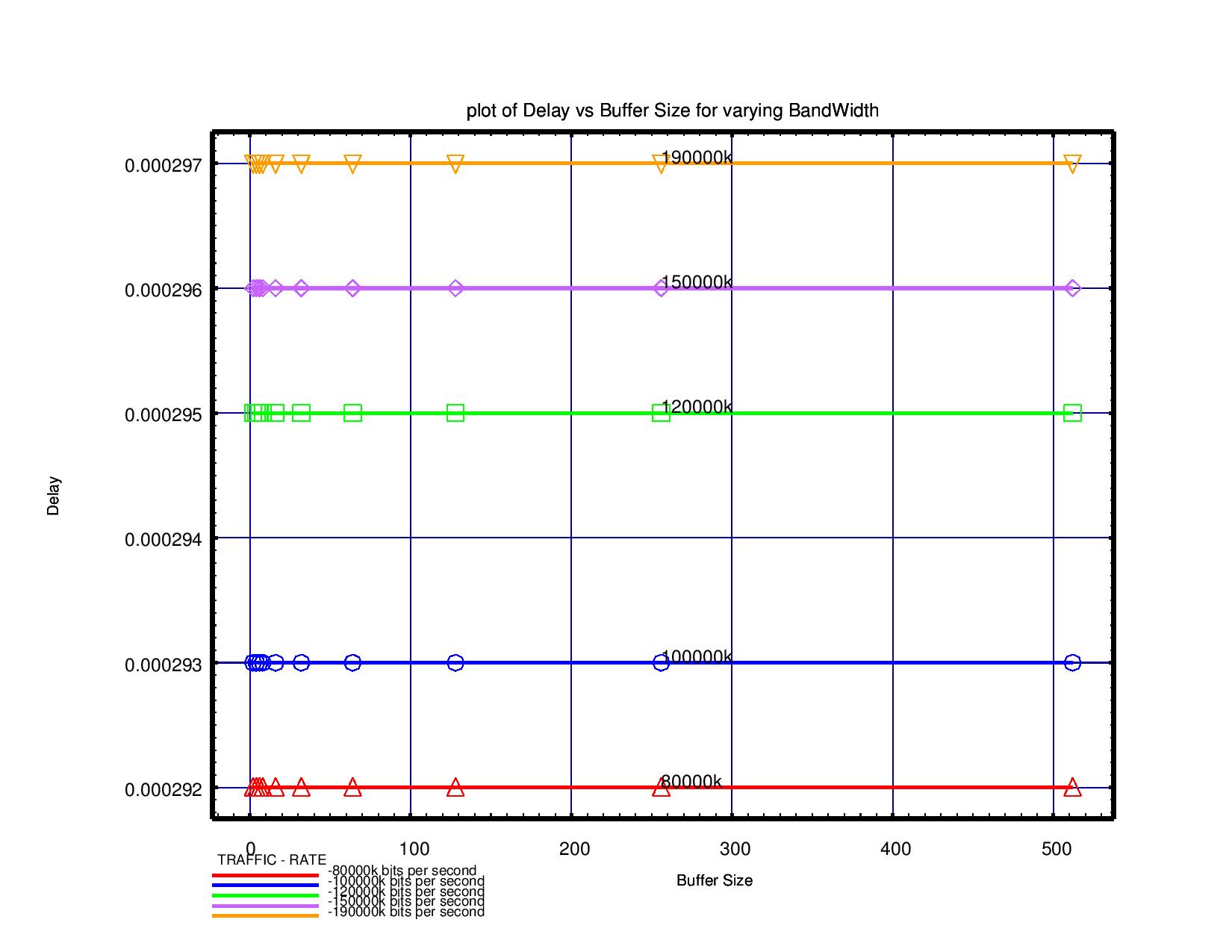
Buffer Size v/s Buffer Utilization

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Buffer Size | Buffer Utilization for traffic rates | | | | |
| 80000k | 100000k | 120000k | 150000k | 190000k |
| 2 | 0.011968 | 0.018082 | 0.026734 | 0.030757 | 0.038796 |
| 4 | 0.016114 | 0.03153 | 0.052773 | 0.058683 | 0.064481 |
| 6 | 0.015973 | 0.030062 | 0.058111 | 0.062491 | 0.068765 |
| 8 | 0.015891 | 0.029132 | 0.059959 | 0.063993 | 0.070371 |
| 16 | 0.01577 | 0.026171 | 0.062081 | 0.065815 | 0.072676 |
| 32 | 0.015711 | 0.020822 | 0.063044 | 0.066458 | 0.073787 |
| 64 | 0.015677 | 0.018275 | 0.063396 | 0.066747 | 0.074469 |
| 128 | 0.015641 | 0.017031 | 0.063442 | 0.066903 | 0.074532 |
| 256 | 0.01556 | 0.016414 | 0.063082 | 0.065861 | 0.074622 |
| 512 | 0.015374 | 0.016088 | 0.062129 | 0.065465 | 0.074581 |

**5.1.4 Link State Routing**

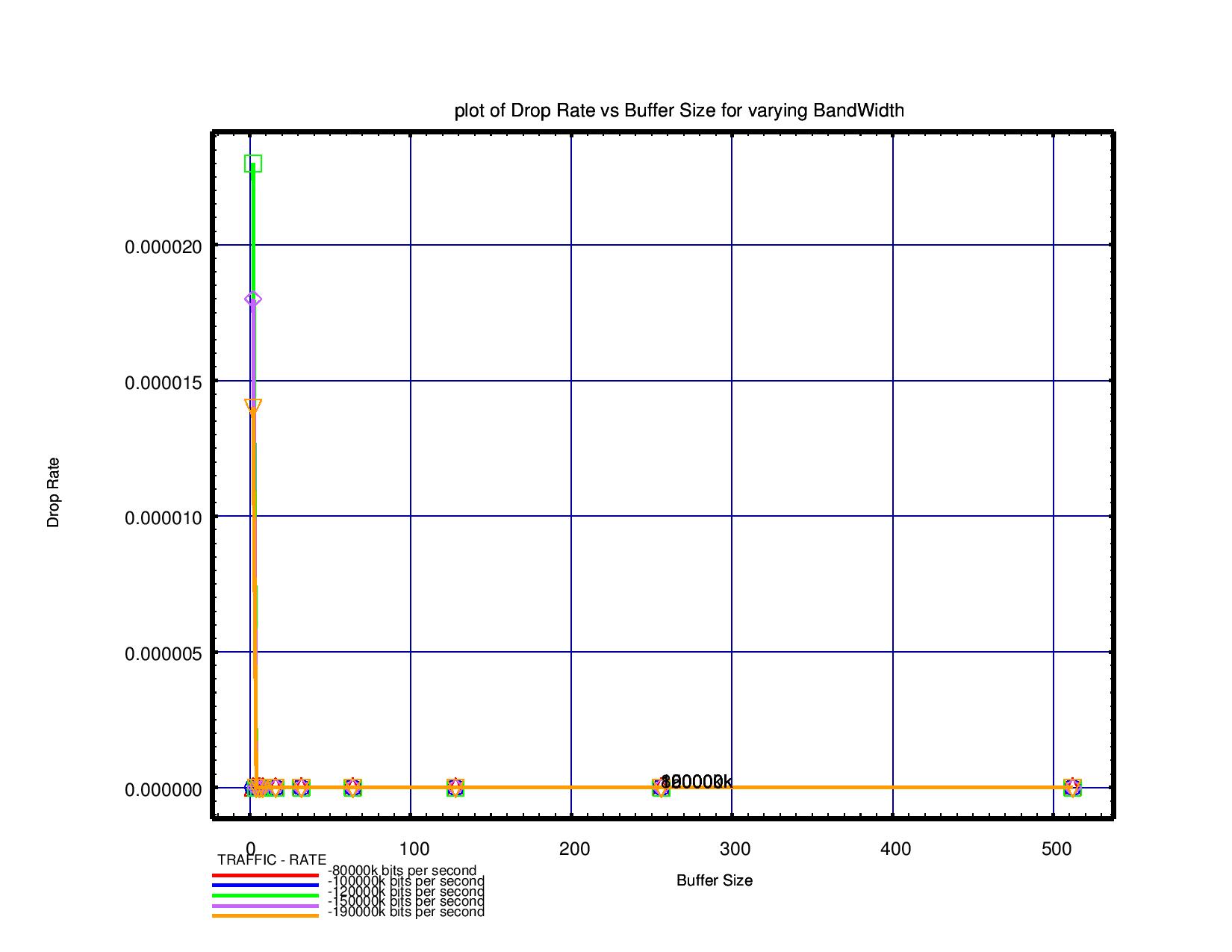
Buffer size v/s End-to-End Delay

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Buffer Size | Delay(micro secs) for traffic rates | | | | |
| 80000k | 100000k | 120000k | 150000k | 190000k |
| 2 | 0.000292 | 0.000293 | 0.000295 | 0.000296 | 0.000297 |
| 4 | 0.000292 | 0.000293 | 0.000295 | 0.000296 | 0.000297 |
| 6 | 0.000292 | 0.000293 | 0.000295 | 0.000296 | 0.000297 |
| 8 | 0.000292 | 0.000293 | 0.000295 | 0.000296 | 0.000297 |
| 16 | 0.000292 | 0.000293 | 0.000295 | 0.000296 | 0.000297 |
| 32 | 0.000292 | 0.000293 | 0.000295 | 0.000296 | 0.000297 |
| 64 | 0.000292 | 0.000293 | 0.000295 | 0.000296 | 0.000297 |
| 128 | 0.000292 | 0.000293 | 0.000295 | 0.000296 | 0.000297 |
| 256 | 0.000292 | 0.000293 | 0.000295 | 0.000296 | 0.000297 |
| 512 | 0.000292 | 0.000293 | 0.000295 | 0.000296 | 0.000297 |



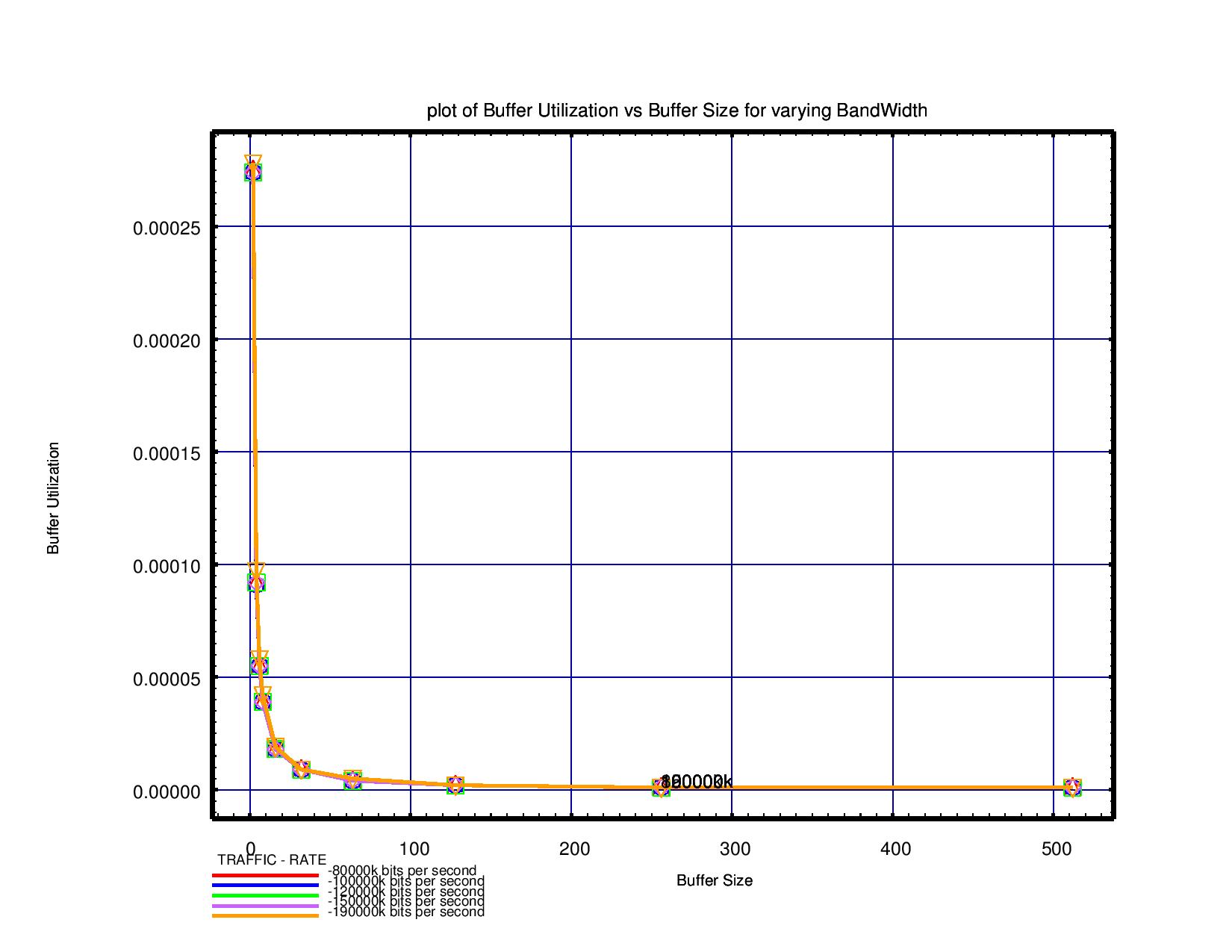
Buffer size v/s Drop Rate

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Buffer Size | Drop Rate for traffic rates | | | | |
| 80000k | 100000k | 120000k | 150000k | 190000k |
| 2 | 0 | 0 | 0.000023 | 0.000018 | 0.000014 |
| 4 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 |
| 16 | 0 | 0 | 0 | 0 | 0 |
| 32 | 0 | 0 | 0 | 0 | 0 |
| 64 | 0 | 0 | 0 | 0 | 0 |
| 128 | 0 | 0 | 0 | 0 | 0 |
| 256 | 0 | 0 | 0 | 0 | 0 |
| 512 | 0 | 0 | 0 | 0 | 0 |



Buffer size v/s Buffer Utilization

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Buffer Size | Buffer Utilization for traffic rates | | | | |
| 80000k | 100000k | 120000k | 150000k | 190000k |
| 2 | 0.000275 | 0.000274 | 0.000274 | 0.000274 | 0.000278 |
| 4 | 0.000092 | 0.000091 | 0.000092 | 0.000092 | 0.000097 |
| 6 | 0.000055 | 0.000055 | 0.000055 | 0.000055 | 0.000058 |
| 8 | 0.000039 | 0.000039 | 0.000039 | 0.000039 | 0.000042 |
| 16 | 0.000018 | 0.000018 | 0.000018 | 0.000018 | 0.000019 |
| 32 | 0.000009 | 0.000009 | 0.000009 | 0.000009 | 0.000009 |
| 64 | 0.000004 | 0.000004 | 0.000004 | 0.000004 | 0.000005 |
| 128 | 0.000002 | 0.000002 | 0.000002 | 0.000002 | 0.000002 |
| 256 | 0.000001 | 0.000001 | 0.000001 | 0.000001 | 0.000001 |
| 512 | 0.000001 | 0.000001 | 0.000001 | 0.000001 | 0.000001 |



**Chapter 6: Findings and Conclusion**

**6.1 Conclusion**

In the following project we have tested four types of routing algorithms on 4\*4 mesh static routing, XY routing, distance vector routing, ant-colony routing. From the results we can derive that delay in case of static, XY and distance vector increases as we increase the buffer size and in ant-colony it remains constant also the delay is minimal among all others. In case of drop rate, it is decreasing in all the three cases except for ant-colony where the drop is zero. Buffer utilisation decreases with the increase in buffer size for all the cases. Therefore, we can conclude that ant-colony routing is the best of all the four since drop rate is zero and also delay is minimal than other three routing plus constant.

**6.2 Future Work**

The future work is to further reduce these parameters and also we will try to test more parameters for NoCs to evaluate the performance.