# Performance Evaluation of Flow Transfer Mode (FTM) with QoS and Comparative Analysis with OBS


**By**

Muhammad Imran

2007-NUST-MS PhD-IT-26

**Thesis Supervisor:**

Dr. Khurram Aziz

**NUST-SEECS**

A thesis submitted in partial fulfillment of the requirements for the degree of Masters of Science in Information Technology (MS IT)

In

School of Electrical Engineering and Computer Science (SEECS),
National University of Sciences and Technology (NUST),
Islamabad, Pakistan.

(April 2011)

# APPROVAL

It is certified that the contents and form of thesis entitled "Performance Evaluation of Flow Transfer Mode (FTM) with QoS and Comparative Analysis with OBS" submitted by Muhammad Imran have been found satisfactory for the requirement of the degree.

Advisor: Dr. Khurram Aziz

Signature: _____

Date: _____

Committee Member1: Dr. S.M.H. Zaidi

Signature _____

Date: _____

Committee Member2: Mr. Muhammad Ramzan

Signature _____

Date: _____

Committee Member3: Ms. Savera Tanvir

Signature _____

Date: _____

# Dedication

Dedicated to dearest of all, my Parents and Teachers
Whose prayers and support stimulate my spirits
to achieve higher ideals of life.

# Certificate of Originality

I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at NUST-SEECS or at any other educational institute, except where due acknowledgement has been made in the thesis. Any contribution made to the research by others, with whom I have worked at NUST SEECS or elsewhere, is explicitly acknowledged in the thesis.

I also declare that the intellectual content of this thesis is the product of my own work, except for the assistance from others in the project's design and conception or in style, presentation and linguistics which has been acknowledged.

Author Name: Muhammad Imran

Signature:

# Acknowledgments

First and foremost, I am immensely thankful to Almighty Allah for letting me pursue and fulfill my dreams. Nothing could have been possible without His blessings.

I would like to thank my parents for their support throughout my educational career They have always supported and encouraged me to do my best in all matters of life.

My heartfelt thanks to my committee members, class fellows and all the others who have contributed in any way towards the successful completion of this thesis.

Finally, this thesis would not have been possible without the expert guidance of my advisor, Dr. Khurram Aziz, who has been a great source of inspiration for me during these years of research. Despite all the assistance provided by Dr. Khurram Aziz and committee members, I alone remain responsible for any errors or omissions which may unwittingly remain.

Muhammad Imran

# Table of Contents

# List of Abbreviations

| | |
|---|---|
| ACK | Acknowledgement |
| ADSL | Asymmetric Digital Subscriber Line |
| ATM | Asynchronous Transfer Mode |
| CP | Control Packet |
| DWDM | Dense Wavelength Division Multiplexing |
| FDL | Fiber Delay Line |
| FTM | Flow Transfer Mode |
| FTTH | Fiber to the Home |
| HP | High Priority |
| IP | Internet Protocol |
| JET | Just Enough Time |
| JIT | Just In Time |
| LAUC | Latest Available Unused Channel |
| LAUC-VF | Latest Available Unused Channel with Void Filling |
| LP | Low Priority |
| Min-SV | Minimum Starting Void |
| Min-EV | Minimum Ending Void |
| MP | Medium Priority |
| NAK | Negative Acknowledgement |
| OBS | Optical Burst Switching |
| OCS | Optical Circuit Switching |
| OPS | Optical Packet Switching |
| OTD | Offset Time Differentiation |
| OXC | Optical Cross-connect |
| O-E-O | Optical to Electrical to Optical |
| PD | Preemptive Dropping |
| P2P | Pear to Pear |
| QoS | Quality of Service |
| RAM | Random Access Memory |
| SDH | Synchronous Digital Hierarchy |
| SONET | Synchronous Optical Networking |
| VoIP | Voice Over IP |
| WDM | Wavelength Division Multiplexing |
| WLAN | Wireless Local Area Network |

# List of Tables

# List of Figures

# Abstract

Demand of multimedia traffic is increasing day by day due to broadband internet access and new end-user business applications as well as the continuing paradigm shift from voice to data services. Multimedia applications require large bandwidth. Optical networks are a good choice for multimedia applications due to huge bandwidth support. Optical Circuit Switching (OCS), Optical Packet Switching (OPS) and Optical Burst Switching (OBS) are some of the switching techniques available in optical networks. All techniques have some limitations and advantages over one another. OCS has round trip delay and bandwidth under-utilization issues. OPS has limitations of unavailability of appropriate optical RAM as well as output port contention, and OBS has problems of burst losses and throughput maximization. Flow Transfer Mode (FTM) is another technique which has tried to address limitations of the existing switching techniques. It is a universal switching method which classifies traffic flows into different modes. Each flow is followed by a control packet that is send in advance just like OBS. Flow can be long in case of continuous or periodic modes and small in case of burst or packet mode. Thus, it is considered as a generalization of OBS. FTM is a generic idea which has not yet been implemented or even tested experimentally. We evaluate the performance of FTM and perform comparative analysis with OBS. For this, we simulate FTM and OBS under different network scenarios. Also we propose and employ QoS provisioning in FTM and evaluate its performance with QoS. Our results show improvement in burst loss ratio, bandwidth utilization and throughput.

# 1. Introduction

## 1.1 Introduction & Background

Demand of multimedia traffic is increasing day by day due to broadband internet access and new end-user business applications as well as the continuing paradigm shift from voice to data services. The applications like peer-to-peer (P2P), data/multimedia, file exchange, video broadcasting, grid services and real time applications are the most bandwidth hungry applications and the trend of usage of these applications is increasing due to immense progress in the deployment of broadband access network technologies (e.g., ADSL, WLAN, or FTTH) which generate huge traffic on the metro and core transport networks. As a result, the next generation future networks should provide high transmission and switching technologies in order to fulfill the requirements of increasing traffic demanding applications. Another trend is increasing due to expansion of the Internet which demands more bandwidth is the shifting of paradigm from voice to data services. This is happening because telecommunication industry is migrating from voice services to IP-centric data services.

So in order to meet future requirements of bandwidth hungry applications, optical network has been considered as the reasonable solution for the next generation future network due to provisioning of high bandwidth and its services. Large bandwidth support has been made possible due to its career i.e optical fiber. In theory, optical fiber can provide bandwidth up to 12.5 Tbits/s [1]. Aside from this, they are cheap and also have low bit loss rate [2] as compared to copper cables.

Figure 1.1 provides a basic architecture of optical network. Optical network consists of source and destination edge nodes and intermediate core nodes which are connected with each other by optical fibers. Client networks (IP, ATM, SONET/SDH, etc.) are connected to the edge nodes. Edge nodes are responsible for conversion of data signal from its input form i.e electrical form to an optical format. The optical format data is transmitted through optical fiber towards core nodes. Several data signals can be traveled at single fiber using Dense Wavelength Division Multiplexing (DWDM). DWDM allocates different wavelengths to different signals. Core nodes are responsible for processing of control information and all-optical switching of data signals. In

most cases the processing of control information is performed electronically. Data is again converted into its client signal format when it arrives at the destination edge node.

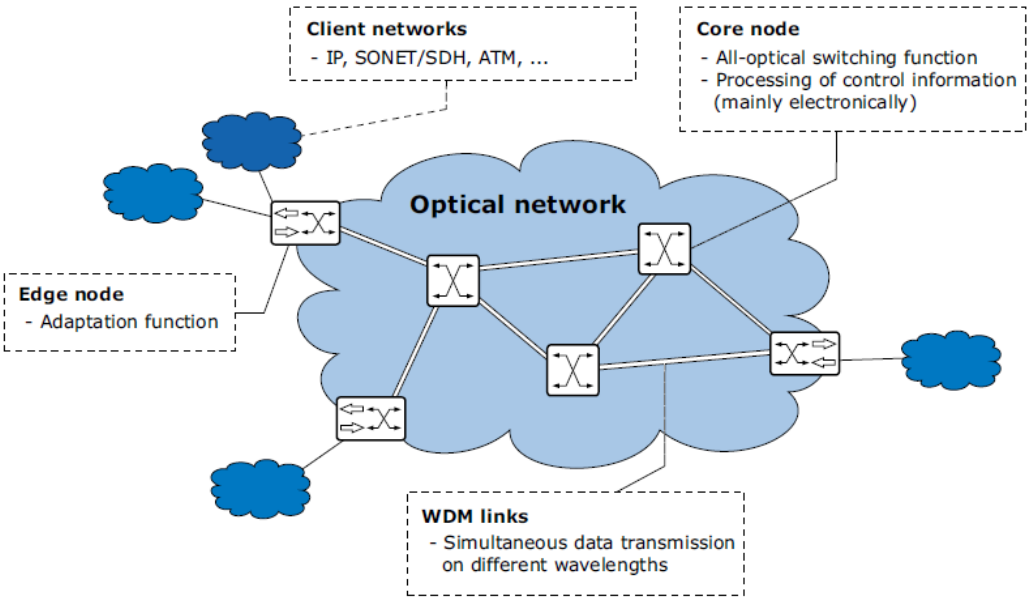



**Figure 1.1 Architecture of DWDM Network [5]**

On the basis of network architecture of optical network, a few network architectures exists i.e. Optical Circuit Switching (OCS) [5,6,7], Optical Packet Switching (OPS) [5,8] and Optical Burst [3,4] Switching (OBS). All techniques have some limitations and advantages over one another.

Optical Circuit Switching (OCS) is connection oriented optical switching technique. In OCS, before data transmission, the connection is established on pre-defined paths from a sending identity to receiving identity which is called light-path. Light-path is a dedicated connection between sender and receiver for the transmission of data. Therefore OCS has issues of round trip delay due to connection establishment and bandwidth under-utilization due to dedicated bandwidth reservation.

OPS is same as of packet switching. In OPS, packet comprises two parts i.e. data and header. Both header and data in packet are in optical domain. When packet arrives at core node, the header is separated from the packet and is converted into electrical domain. During this time, packet has to be buffer in core node. If output port is not free, then packet will have to be buffer

or ultimately will drop. Since the technology of optical buffers is not very much mature therefore OPS has limitations of unavailability of appropriate optical RAM and output port contention.

OBS technology can be seen as a promising solution. OBS differs from other paradigms because control information is sent via control packets in advance of the data payload which are called bursts. Reserved optical channel is dedicated for control packets. These control packets are then processed electronically to reserve channel for burst which come after offset time. But OBS also has some of the issues like burst losses, unfairness in access to transmission resources, complexity of control and throughput maximization.

In order to address limitations in above technologies a universal switching method named as Flow Transfer Modes (FTM) was proposed in [9-13]. Just like OBS, FTM also consists of control plane and data plane. Control plane is kept for control packets and data plane is reserved for data to be sent from source to destination. Basic difference between FTM and OBS is that FTM classify traffic according to different data flows which are called modes. It provides 4 different modes i.e. Continuous Streaming Mode, Periodic Streaming Mode, Burst Mode and Packet Mode. Data transmissions of each mode start after control packet which is sent a fixed time before actual data transmission which is called offset time.

The author in [9-13] has given generic idea for FTM which will likely to be used as universal switching technique in all optical future networks. In this thesis, we evaluate the performance of FTM and perform comparative analysis with OBS. For this, we simulate FTM and OBS under same simulation parameters and different network scenarios. Then we also propose and employ QoS provisioning in FTM because bandwidth hungry applications generally require QoS provisioning. We evaluate performance of FTM with QoS. Our results show improvement in burst loss ratio, bandwidth utilization and throughput.

## 1.2 Contribution

This thesis provides the first implementation of FTM and provision of QoS in FTM according to the best of authors' knowledge. We have done comparative analysis with OBS and our results show that FTM shows better performance than OBS. Also by provisioning of QoS, FTM performance is further improved.

### 1.2.1 Problem Statement

*"To propose quality of service mechanism in flow transfer mode, evaluate its performance and perform comparative analysis with OBS."*

### 1.2.1 Problem Breakdown

The problem is broken down into following steps to achieve specific objectives:

- *Implementation of OBS:* First of all OBS network is implemented using simulation by on the basis of network design and simulation parameters that we assumed before simulation. Then results of OBS are compared with existing implementations of OBS in order to assure simulation is correct.
- *Implementation of FTM:* After OBS implementation, FTM is implemented using same network design and simulation parameters in OBS.
- *QoS Provisioning in FTM:* The objective of this step is to propose QoS provisioning mechanisms in FTM and implement FTM using same simulation parameters and network design.
- *Performance Evaluation:* In this step, comparative analysis and performance evaluation of FTM with OBS and then with QoS provisioning in FTM has been done.

## 1.3 Thesis Organization

The remaining part of this thesis is arranged as follows:

In chapter 2, a detailed discussion on some of the existing optical switching techniques i.e. OCS, OPS, OBS has been provided. Their functionality, advantages, limitations and comparison is being done in this chapter. In the end different QoS employment techniques in OBS is described

Chapter 3 provides detailed description of FTM, i.e. different modes of FTM, current and future networking architectures and FTM layered architecture.

In Chapter 4, proposed mechanism of QoS in FTM is presented briefly.

In Chapter 5, Implementation techniques of OBS, FTM, and FTM with QoS is discussed. Then their performance is evaluated and comparative analysis has been done. Limitations of the proposed technique are also discussed.

In Chapter 6, we briefly summarize this work and discuss future direction of this thesis.

# 2. Literature Review

In this chapter, a detailed discussion on some of the existing optical switching techniques has been provided. In optical network, there are three switching techniques which are used to achieve better performance at very high data rates. There are classified as:

- Optical circuit switching (OCS)
- Optical packet switching (OPS)
- Optical burst switching (OBS)

## 2.1 Optical Circuit Switching:

Optical Circuit Switching (OCS) is connection oriented optical switching technique. In OCS, before data transmission, the connection is established on pre-defined paths from a sending identity to receiving identity which is called light-path and is shown in the figure2.1. Light-path is dedicated connection between sender and receiver for the data transmission.

OCS switching nodes are called optical cross-connects (OXC). All optical switching is provided by an OXC. OXC has input and output ports. OXC receives data at incoming port by using a particular wavelength and forwards it to an outgoing port to the same wavelength. If same output wavelength is not available or free then wavelength converters can be employed. Wavelength converters convert input wavelength to output wavelength.[5]

To establish the connection, control packet is send from source to destination and then back to source which carries control information for the connection establishment and it is considered an overhead. Typical connection durations are expected to be even as low as some seconds and the connection setup and release can be performed during some ms [5].

There are two major issues which is associated with OCS.

- Complete round trip delay
- Bandwidth under utilization

Complete round trip delay occurs before data transmission during connection establishment. It depends on the transmission channel capacity and distance from sources to destinations. Second issue is related with under utilization of bandwidth. Because connection is dedicated for source to destination and when source does not send data then bandwidth is under utilized.

Contention can be occurred if there are not enough wavelengths for the incoming request. In result of this, request may be delayed, blocked or dropped.

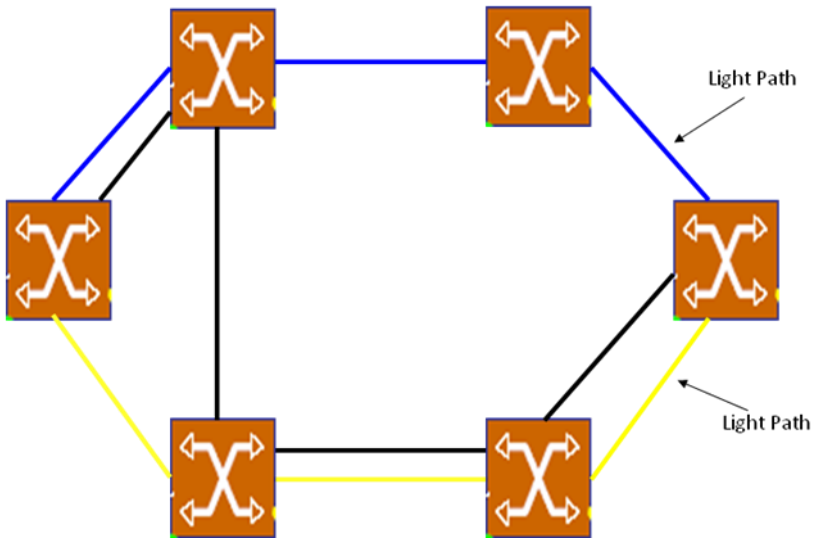



**Figure 2.1: Optical Circuit switching [14]**

## 2.2 Optical Packet Switching:

The concept of Optical Packet Switching (OPS) is same as of packet switching. In packet switching, data is divided into different size of chunks which are called packets. Packets are delivered to the next-door router. The routing mechanism is performed on hop-by-hop basis. Routing decision is carried on the basis of destination address independently and irrespective of the routing decisions in other routers. The network also does not maintain state of the packet other than the routing tables [14].

The routers do forwarding on the basis of store-and-forward mechanism in which IP packets are stored temporarily in the RAM of routers during processing and then transmitted. During

network congestion, packets may also need to be stored locally and if the system undergoes short of memory, then packets are discarded.

In OPS, packet comprises two parts i.e. data and header. Both header and data in packet are in optical domain as shown in Figure 2.2. When packet arrives at core node, the header is separated from the packet and is converted into electrical domain. Core node does processing for the next hop and then converts header again into optical domain and merge with payload again. During processing time payload has to wait at the core node. In this way, header goes through optical to electrical to optical (O-E-O) conversion. Wavelength resources are shared between packets belonging to different transmission. Concept of an entirely optical OPS router is also available which is supposed to process this control information in an optical way but due to still immature all-optical processing the header is usually converted to its electrical form and processed in an electronic node controller [14].

Processing in OPS is done in three steps. In first step, packet arrives at core node which is shown in the figure2.2. In step 2, header is separated from the packet and is converted into electrical domain for processing as shown in the figure2.3. In step 3, after processing header is again converted into optical domain and is merged with the payload for transmission as shown in the figure2.4.[14]


**Figure 2.2: Data arrives at the core node [14]**

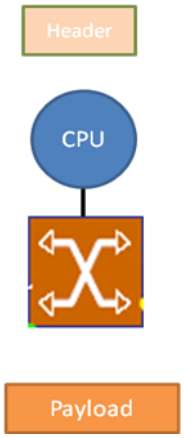



**Figure 2.3: Processing at core node [14]**


**Figure 2.4: Data transmitted by core node [14]**

There are three main issues which are concerned in OPS.

- Lack of optical  RAM (Random Access Memory)
- Output port contention
- Processing overhead at each node

In first issue, optical RAM is not available. For limited buffering, fiber delay lines (FDL) can be employed as alternative but it is costly solution and is not viable and effective as well because of limited time storage.

Second issue is related with output port contention. If network is congested, then packet will be dropped.

Third issue is related with processing overhead at each node. For example, during transmission of a video stream, every packet will have to be individually scheduled and processed at each node which ultimately incorporates more delay.

## 2.3 Optical Burst Switching:

Optical burst switching (OBS) [3,4] is a an emerging technique that is considered a more appropriate solution than optical packet switching (OPS) and optical circuit switching (OCS). OBS basically unites the features of OCS and OPS. In OBS neither circuit is established as in OCS or OEO conversion for the data is required at core nodes as in OPS.

OBS differs from other techniques in such a way that control information is sent via control packets in advance of the data payload which are called bursts. Control packets are processed in electrical domain for reservation of channel for burst which comes after offset time [15]. Reserved optical channel is dedicated for control packets.

Similar to OPS, the wavelength resources are shared between different connections in OBS. OBS network in shown in the figure2.5. Packets arrive at the source edge nodes of an OBS network from legacy networks (e.g., IP, ATM networks) and are aggregated into large optical bursts. For time duration equal to offset time, these bursts are temporarily stored at edge node before transmission. Before burst transmission, a control packet is send. The control packet and burst are transmitted separately on reserved wavelengths.

Burst is send after particular time of control packet which is called offset time. In offset time, controller of the core node processes the control information and setups the switching matrix for the incoming burst. The burst travels already configured nodes all the way in optical domain. The duration of typical burst, which aggregates a group of packets, can last from some micro seconds to several hundreds of milli seconds depending upon burst size and transmission line capacity.

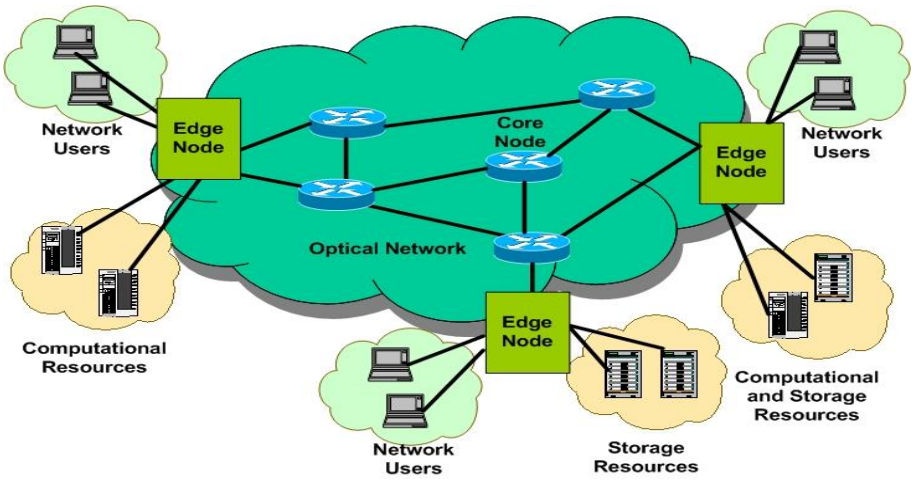



**Figure 2.5: OBS Network [14]**

Partition of data and control plane is the main characteristics of OBS network. O-E-O conversions takes place in case of control packets while in there is no O-E-O conversions i.e. data travels all optically. Because of the absence of O-E-O conversion in data plane delay is reduced to a large extent.

## 2.3.1 Burst Aggregation:

In OBS network, packets which arrive at the source edge node are combined to form a burst on the basis of destination. Formation of burst from the incoming packets is known as burst aggregation. [4]

Following are the types of aggregation algorithm [16]:

#### *2.3.1.1 Timer-Based algorithm:*

In this algorithm, burst is formed after a particular fixed time. A timer starts from the beginning and after a predefined time, the packets which have come in edge node within that timeframe are combined to form a burst. Timer should not be small or large for example in case of short timer there will be more overheads in terms of control packets and in case of large timer there will be more delay.

#### *2.3.1.2 Burst-Length Based algorithm:*

In this algorithm burst is formed when packets in a destination queue of edge node are reached to a certain threshold value for burst length. Threshold length value should not be small or too large because if it is kept small, then there will be more overheads in terms of control packets and if it is large then large amount of data can be lost in case of burst loss.

#### *2.3.1.3 Hybrid Approach:*

It is a combined approach to overcome inefficiencies which are present with the above techniques. In this technique, the threshold value for burst length is reached or timer is finished then the burst is formed.

#### *2.3.1.4 Adaptive Assembly based algorithm:*

This algorithm is same as of hybrid approach but parameters of both time and threshold length is dynamically adjusted.

### 2.3.2 Burst Reservation Protocols:

Burst reservation protocols are of two types [4].

- One way signaling
- Two way signaling

Two way signaling protocols also called Tell and Wait signaling protocol. In this technique source sends a control packet from source to destination for wavelength reservation. Control packet is passed through intermediate nodes. Every intermediate node processes this control packet and reserves a wavelength for incoming burst. If the wavelength is successfully allocated on the path all the way through, then an acknowledgment (ACK) is sent back to sender and sender sends burst immediately; If channel is not reserved then a negative acknowledgment (NAK) is send to free the wavelength which was reserved earlier and retransmit again by sending control packet again after back off time. These algorithms focus on minimal burst losses. They are less efficient because of complete round trip delay which is occurred before transmission of burst.

One way signaling protocols are also named as Tell and Go protocols. The conventional OBS uses one way signaling protocol. In one way signaling protocol, after offset time burst is transmitted without waiting for acknowledgement. But in these protocols, there is a chance of burst loss as the switch may be blocked.

Following are the two important protocols for practical implementation of OBS. Though there are other mechanisms available for signaling but they are derived from the following two.

- Just in Time (JIT)
- Just enough Time (JET)

#### *2.3.2.1 Just In Time (JIT) Protocol:*

Just-In-Time (JIT), was proposed in [17], and is based on tell-and-wait technique with slight variations. In JIT, control packet is sent to a central node which is responsible for scheduling purposes. The central node then notifies all the respective nodes the exact time of the burst transmission. Another variation of JIT which is called Reservation with Just-In-Time was proposed in [18] to incorporate more scalability and robustness in JIT. In this algorithm, copy of the request is sent to all nodes simultaneously. The schedulers exchange information of link status and are synchronized in time with each other.

#### *2.3.2.2 Just Enough Time (JET) Protocol:*

JET [3,21] is the most commonly used protocol in OBS networks. Optical buffering or any type of delay is not required at core nodes. This happens due to control packet which carries offset time information. In JET, total burst reservation time is equal to arrival time of the burst and actual duration of burst. In JET, the bandwidth is reserved from the burst's incoming time regardless of incoming time of control packet which was the case in JIT. Offset time is reduced with respect to the processing time at all intermediate core nodes [4].

### 2.3.3 Scheduling Algorithms

By assuming wavelength conversion using wavelength converter, an incoming burst can be scheduled at any output wavelength on required port. Scheduling algorithm will decide which

wavelength to be scheduled for output port. There are some scheduling algorithms exist and is shown in the figure2.6.

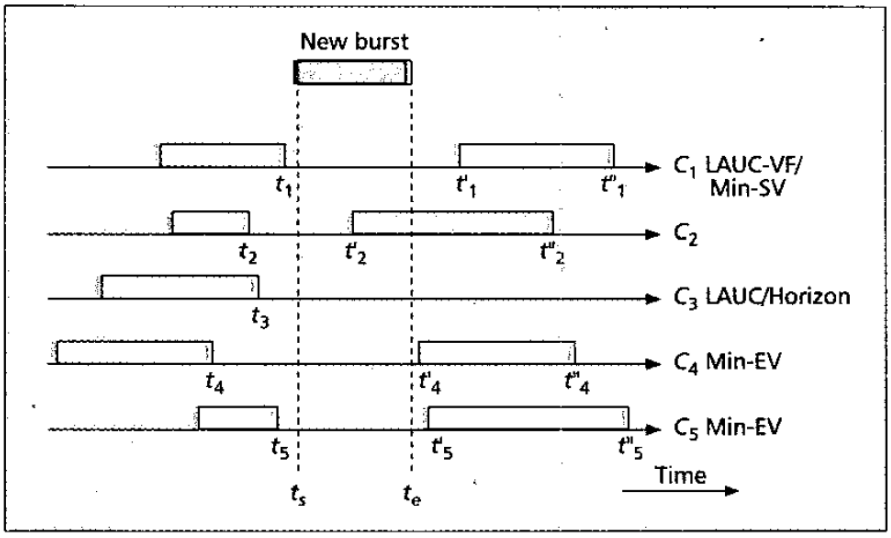



**Figure2.6: An Illustration of Scheduling Algorithms [4]**

The latest availabe time at which the channel can be schedule is known as scheduling horizon. In figure. 2.6, for example, time t1,t2,t3,t4,t5 are the scheduling horizons for channel C1, C2, C3, C4 and C5 respectiverly.

Consider a simple algorithm, latest available unscheduled channel (LAUC) algorithm in [19]. In this algorithm, a single horizon is retained for each wavelength. Only channels are considered available which has less horizons than incoming burst's arrival time and the channel who has latest horizon is selected. By using this technique, C3 will be selected as shown in the figure 2.6. LAUC has advantage of simplicity and implementation but gaps between two already configured wavelength are wasted. Therefore void filling algorithms were introduced to avoid this problem.

Void filling algorithms can make new resewations within existing gaps. Latest Availabe Unused Channel with void filling (LAUC-VF), was presented in [19]. Due to capability of scheduling burst between already scheduled bursts, channel Cl will be selected. In this way the gap which was wasted in LAUC is utilized by uing LAUC-VG algorithm. Then few variations in LAUC-VF algorithm were proposed in [20] i.e. Min-SV (Starting Void), Min-EV(Ending Void) and Best Fit. Min-SV is theoretically similar to LAUC-VF but it is implemented using a different

method in [20]. While Min-EV tries to decrease the new gap produced between already scheduled burst's incoming time and incoming burst's ending time, while Best Fit is based on the minimum of both of the length of starting and ending voids. These scheduling techniques and there outcomes is presented in the figure2.6. The comparative analysis of these techniques in [20], illustrates that all void filling algorithms have higher link utilization and less burst loss ratio as compared to Horizon-based algorithms without void filling[4].

### 2.3.4 Contention Resolution:

Contention can be occurred if no wavelength is available at desire output port for incoming bursts. If contention is not resolved then incoming burst is dropped. Following are few of the techniques which are used to avoid contentions in OBS network [14].

- Fiber Delay Lines (FDL),
- Deflection Routing,
- Wavelength Conversion.
- Segmentation based dropping

Fiber Delay Lines (FDL) can be used to provide temporary storage of bursts but this technique is not very efficient due to expensive FDL lines.

Deflection Routing is another technique in which incoming conflicting burst is tried to schedule using another physical path to the same destination. This approach can be used when network size is reasonably high i.e. there are more physical paths available between two edge nodes.

In wavelength conversion, wavelength converters are used in order to convert one wavelength to another. When there are two bursts destined for same output port, the core node assigns a different wavelength to the incoming burst. Although wavelength converters are expensive but many scheduling algorithms assume full wavelength converters.

Segmentation based dropping is another technique in which only the segment which overlaps with existing burst is dropped.

### 2.3.5 Quality of Service in OBS

Quality of service (QoS) mechanisms in a network can be classified into two types [14]:

1) QoS provisioning mechanisms
2) QoS improvement mechanisms

*QoS provisioning* in a network is further divided into two types i.e.

1) Relative QoS provisioning

2) Absolute QoS provisioning.

With relative QoS provisioning, traffic is divided into difference classes. Some classes are assigned high priority over other classes and then performance of classes are measured and compared with other classes. In this way, higher priority classes show better performance as compared to lower priority classes.

The absolute QoS model in OBS assumes to give quantitative loss guarantees to traffic classes in worst-case scenarios as well. Absolute QoS mechanisms discriminates classes on the basis of absolute threshold values.

*QoS improvement mechanism* is defined as any technique which progresses the overall network performance. These mechanisms are very important to provide desired services to the end users.

As in Survery of QoS in OBS in [22], QoS can be employed as:

- QoS differentiation with one-way signaling
- QoS differentiation with two-way signaling
- QoS differentiation with control plane methods

Figure 2.7 shows different mechanisms for QoS provisioning in OBS network. There are two different data plane and control plane in OBS network. In data plane some of the techniques use one way signaling while rest of them uses two way signaling protocol methods.

In control plane there are two techniques which can provide QoS mechanism in OBS as shown in the fig. In Signaling technique we can use control packets to provide QoS provisioning. In routing technique we can use deflection routing to provide QoS provisioning.

Data plane is further classified into core nodes and edge nodes. Some of the techniques can be applied at edge nodes and rest of them will be applied at core nodes.

In edge nodes there are two most promising technique that can provide QoS provisioning as shown in the figure 2.7. In offset time based mechanism, higher priority classes will assign additional offset time as contrast to lower priority class. In this way higher priority class burst will have higher chances of burst schedule as compared to lower priority class due to early reservations by increasing offset time. In varying burst assembly parameters, higher priority burst will be schedule faster as compared to lower priority burst.

At core nodes, QoS provisioning can further be divided into two categories i.e burst dropping schemes and differentiation of control packets.

In burst dropping schemes there are three most popular techniques i.e. pre-emptive dropping, threshold based dropping and intentional burst dropping.

In pre emptive dropping when a conflict occurs between bursts of two priority class, then higher priority class burst will be schedule and lower priority class burst will be dropped.

In threshold based dropping, we can assign threshold values e.g. wavelengths to certain class so that this class cannot use more resources as is decided. If threshold reaches then burst will be dropped.

In intentional based dropping, low priority class burst will be intentionally dropped on the basis of certain parameters.

And in the last we can provide QoS provisioning by using scheduling algorithm in control packets.

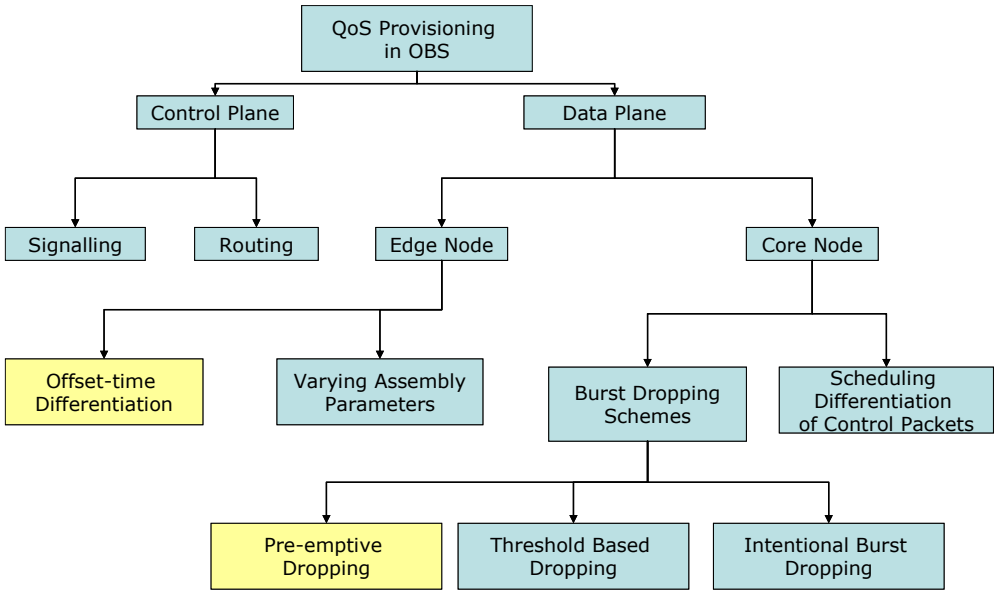



**Figure 2.7 QoS Mechanisms in OBS**

#### *2.3.5.1 QoS Performance Metrics*

There are two major QoS metrics that are used to provide QoS provisioning in OBS. There are :

- Burst Losses
- Delay

In most of them burst losses is the main parameter as shown in the figure 2.1. All techniques support burst losses and few of them support delay as well with burst losses. Table 2.1 shows comparison of different QoS techniques with supported parameter and advantages/disadvantages.

### 2.3.6 Comparisons of QoS mechanisms in OBS.

Brief comparisons of QoS mechanisms in OBS network is described in [22] and is shown in the table 2.1 below.

| QoS mechanism | Implemented QoS model | Supported QoS parameter | Advantages | Disadvantages |
|---|---|---|---|---|
| **Offset time-based QoS differentiation** | relative | burst losses | - simple, soft operation<br>- no need for any differentiation mechanism in core nodes | - sensitivity of HP class to burst length characteristics<br>- extended pre-transmission delay |
| **Burst length-based QoS differentiation** | relative | delay/burst losses | - assembly parameters can be easily setup | - resulting traffic characteristics may influence network performance<br>- requires complex void filling algorithms |
| **Preemptive dropping** | relative/absolute | burst losses | - fine class isolation<br>- improved link utilization in schemes with partial preemption<br>- absolute QoS can be achieved with a probabilistic preemptive scheme | - overbooking of resources in consecutive nodes (in case of successful preemption)<br>- additional complexity involved in burst assembly process in case of partial preemption |
| **Threshold-based dropping** | relative | bursts losses | - can be easily implemented | - efficiency of bandwidth usage strongly depends on threshold adaptability to traffic changes |
| **Intentional burst dropping** | absolute | burst losses | - can provide absolute QoS | - link utilization may suffer<br>- complex implementation |
| **Scheduling differentiation of burst control packets** | relative | burst losses | - priority queuing in electrical buffers is a feasible and well- studied technique | - extended delay (need for longer queuing windows and thus larger offset times to perform effectively) |
| **Differentiated available bit rate** | relative | burst losses | - class isolation achieved<br>- more complex priority models than strict priority can also be enforced | - requires a new rate control protocol and advanced schedulers at the edge for burst shaping |
| **Efficient burst reservation protocol** | relative | burst losses | - class isolation achieved | - requires a complex two-way reservation protocol |
| **Hybrid signaling** | absolute | delay/ burst losses | - absolute end-to-end loss and delay guarantees for HP | - lower statistical multiplexing gain, inefficient usage of bandwidth (fewer resources available for LP traffic) |
| **QoS routing** | absolute (delays)<br>relative (burst losses) | delay/ burst losses | - introduces QoS guarantees at network level | - controlling burst losses may be challenging (need knowledge about network state) |

**Table 2-1: Comparison of QoS Mechanisms in OBS [22]**

# 3. Flow Transfer Mode (FTM)

## 3.1 Introduction

In literature review we have described different switching techniques which are used in OBS network. Every technique has some advantages and disadvantages over one another. For example OCS has round trip delay and bandwidth under-utilization issue. OPS has limitations of unavailability of appropriate optical RAM as well as output port contention, and OBS has problem of burst losses and throughput maximization.

In order to overcome and address limitations in all technique a generic technique named as Flow Transfer Mode (FTM) was proposed in [9-13].

FTM is defined as:

*"FTM is a universal switching method with a layer-1 switching technology, and layer-2 or layer-3 control for scheduling continuous or periodic data flows as well as short flows consisting of a single packet or a burst of aggregated packets. Each flow is triggered by a control packet that is transmitted due time in advance just like OBS. Thus, it is regarded as a generalization of OBS". [12]*

The author in [9-13] has given generic idea for FTM which will likely to be used as universal switching technique in all optical future networks.

Just like OBS, FTM has also consists of data plane and control plane. Control plane is reserved for transmission of control packets and data plane is reserved for data to be sent from source to destination.

## 3.2 Modes of FTM

Basic difference between FTM and OBS is that FTM classify traffic according to different data flows which are called modes. It provides 4 different modes i.e.

- Continuous Streaming Mode.

- Periodic Streaming Mode.
- Burst Mode.
- Packet Mode.

Each mode is followed by a control packet that is send particular time (offset time) in advance.

From networking point of view, layer 1 is used for switching technique and layer 2 or 3 is used for reservation of different data flows. Data flows are small in case of packet or burst mode but it can be very large for continuous and periodic streaming modes. Flows are established on the basis of destination edge nodes similar to OBS. Thus it is regarded as generalization of OBS.

Four modes of flow transfer modes as described in the figure 3.1.

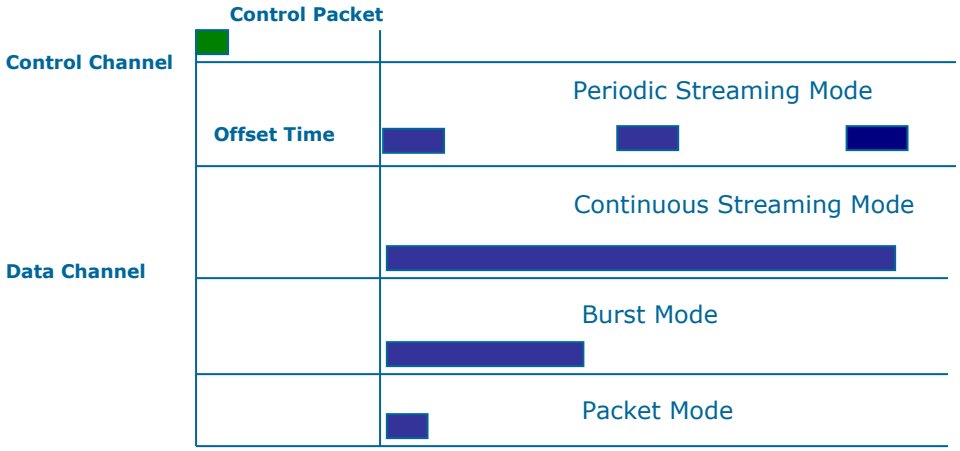



**Figure 3.1 Modes of FTM**

### 3.2.1 Packet mode:

Processing overhead in packet mode is same as of OPS, because number of data packets are same as of number of control packets but network performance can still be improve due to rearrangement / rescheduling of packets before packets actually arrive at core node.

### 3.2.2 Burst mode:

As compared to packet mode, burst mode decreases processing overhead due to creation of bursts. Burst generation can be done either with a length or time based or mixed. This mode is similar to OBS.

### 3.2.3 Periodic streaming mode:

Periodic streaming is just like conventional circuit-switching technique. It requires a single control packet which travels from sender to receiver for wavelength reservation. This mode can be used for the sender which frequently generates a certain amount of data, e.g. VoIP etc.

### 3.2.4 Continuous streaming mode:

This mode is used when high volume of data is to be transmitted among two end systems. For example for applications which require huge bandwidth i.e. real time applications or multimedia applications this mode can be used. This mode can also be used when huge amount of data is to be transmitted between two end systems.

## 3.3 Current and Future Networks Architecture

The communications network architecture can easily be described as on the basis of functionality in which network is divided into layer model like OSI reference model. There are two major areas to be considered in this layer model i.e. transport and network. Current and future states of networking are shown in the figure 3.2 and 3.3. Switching is central plane in this network architecture which also contains all the networking components of all switching technologies.

In this switching plane all equipments and communication media can use electrical or optical plane. This is where FTM comes into play. With current networking state signaling and switching is being done on electrical domain but with the use of FTM, electrical domain will only be reserved for signaling domain and all of the switching will be done in optical domain.

Fig. 3.2 and 3.3 shows the evolution towards FTM which integrates circuit, burst, and packet switching into one single generic method. [12].

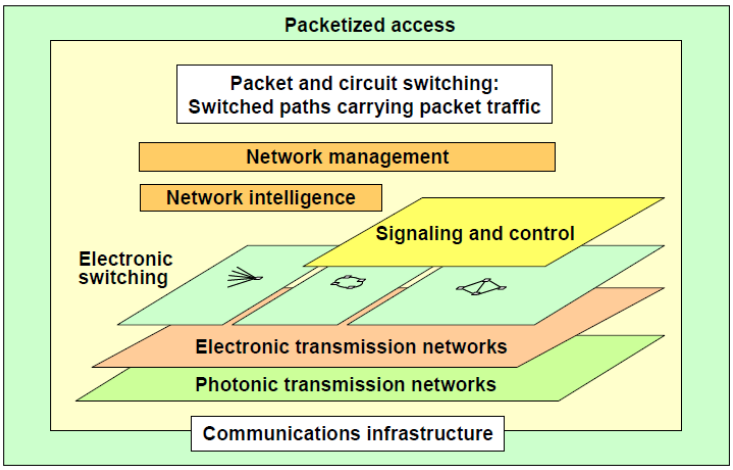



**Figure 3.2: Current layered network architecture [12]**

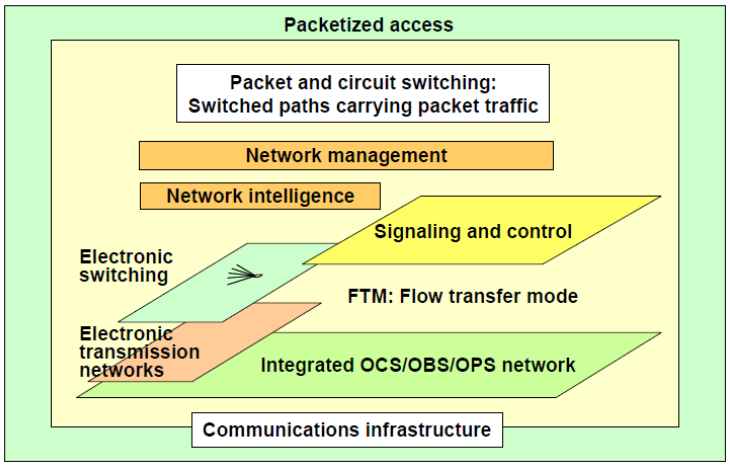



**Figure 3.3: Future layered network architecture [12]**

## 3.4 FTM Layered Network Architecture

FTM layered architecture is divided into five layered according to functionality and is shown in the figure 3.4. Forwarding and forwarding control layers are the bases of this architecture and the three upper layers depict latest communication technologies including protocol and whole networking systems.

1) Service delivery and network intelligence: It is the upper most layer of FTM layered architecture. This layer includes all the protocols, technologies and services through which human interact with devices

2) Signaling plane: To start communication between two end users, end to end connection is required. This is done in this section.

3) Virtual topologies: This layer consists of protocols for paths establishment, routing mechanisms and technique for network resilience which are used to build network topologies.

4) Control plane: Source sends a control packet before sending actual data which travels along the path which are established by virtual topologies all the way from sender to receiver. Control plane transmission is kept on different wavelengths or fiber from the data plane because in data plane, data units are scheduled with inspection using already configured wavelength.

5) Forwarding plane: This layer only provides forwarding / switching of data. The devices which are used in forwarding plane are switches, amplifiers, wavelength converters and FDL.

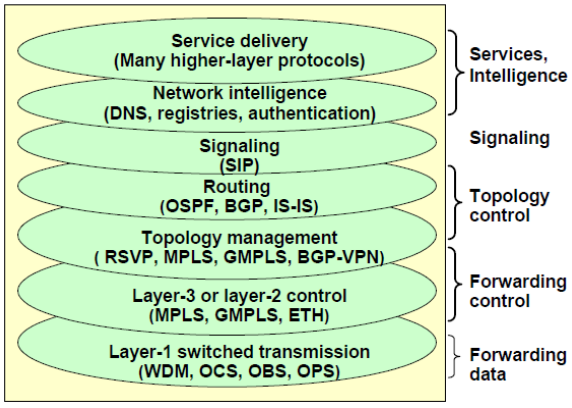



**Figure 3.4: FTM Layered Architecture [12]**

## 3.5 Node architectures in FTM

*Edge nodes:* Data preparation is the main task of edge node. It is responsible for the preparations of different data flows according to requirement. Each data flow is followed by a control packet. Packets are maintained in destination queues for burst mode, and then all the packets in a queue are combined to make a single burst similar to OBS as shown in the figure3.5.

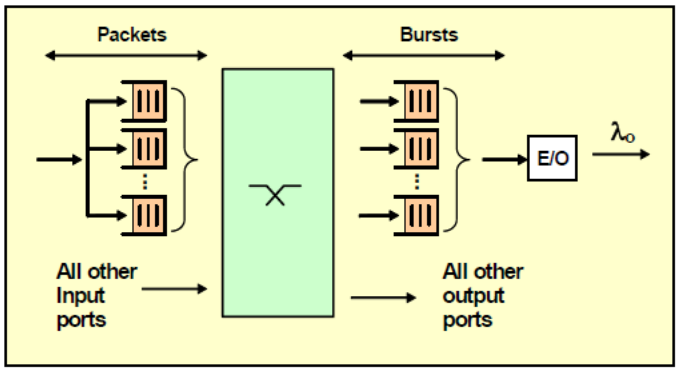



**Figure 3.5: Edge Node Architecture [12]**

*Core nodes:* A core node is shown in the figure 3.6. Core node consists of a scheduling, control and a data switching part. Scheduling is done in electrical domain and forwarding is done in optical domain. Control part is responsible to control overall switching mechanism. E.g. if conflicts occurs then wavelength conversion is done using wavelength converters or delay lines are used temporarily.

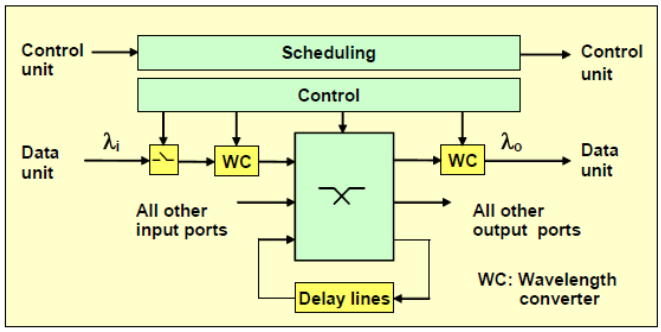



**Figure 3.6: Core Node Architecture [12]**

## 3.6 Operational Issues in FTM

*Time synchronization*: Time synchronization between nodes is required so that nodes can forward optical data without any assessment. Optical data flows are separated by small time so that signals cannot overlap with each other and this time is called guard bands

*Flow notification:* The control packet which is send with an offset time. Offset time also contains total propagation delay, processing delay and switching delay along the path in all the nodes. All

the nodes should have information regarding this delay. It can be done by using a special registry for flow notification and this information can be exchanged with DNS.

## 3.7 Advantages of FTM Over OBS

Biggest advantage of FTM lies in case of multimedia traffic. Streaming modes of FTM seems to provide better performance than OBS.

For example, consider a case of video transfer.

- FTM will generate only 1 control packet but OBS will generate multiple control packets depending upon size of burst and video which results in large amount of processing and delay in OBS.
- Considerable amount of time will also be spent in OBS edge nodes processing in creation of burst assembly and de-assembly and also transmission of burst separately.

In the end we can say that FTM is a universal switching mechanism. FTM combines all switching mechanisms in one single technique. FTM classifies traffic to different modes. The applications decide which mode should be used for incoming traffic. The big advantage of FTM seems to be lie in case of multimedia streaming traffic due to constant network end to end delay.

# 4. Proposed QoS Provisioning in FTM

Author in [4-8] has only given the idea of FTM. It has not been tested experimentally or evaluated by using simulation or any other technique. We have implemented FTM as generalization of OBS so that we can compare FTM with OBS under similar parameters. We have done performance evaluation of FTM by using simulation and then perform comparative analysis of FTM with OBS.

In Offset Time Differentiation (OTD), an extra/additional offset time is allocated to high priority (HP) class which results in prior reservation for HP bursts as compared to low priority (LP) bursts. In this way HP bursts have higher probability of being reserved as compared to LP burst. Question arises about the length of extra offset time. Length of the extra offset time should not be less than mean burst size of low priority bursts in order to achieve perfect isolation of HP and LP classes.

In Preemptive dropping (PD): when a request for high priority class burst comes at the core node and wavelength in the output port can not be reserved, then the wavelength which is already reserved an LP burst is assigned to overlapping HP burst and LP burst is dropped. There are two variations of preemptive dropping technique, i.e. full preemption and partial preemption. In full preemption whole LP burst is dropped to schedule HP burst while in partial preemption only overlapping part of LP burst is dropped.

We have used full Preemptive Dropping technique with the combination of Offset Time Differentiation. In Pre-emptive Burst dropping technique, there are two classes of traffic. Traffic is divided according to three classes i.e. Class 0, Class 1 and Class 2 and assign priorities to three classes as:

- Class 0 for High Priority.
- Class 1 for Medium Priority.
- Class 2 for Low Priority.

As there are four modes in FTM, but we have used three modes. Packet mode is not considered in our implementation because we have to compare it with OBS and we have implemented it as generalization of OBS. We use different classes for different modes as:

- Class 0 is used for continuous streaming mode.
- Class 1 is used for periodic streaming mode.
- Class 2 is used for burst mode.

Different classes are assigned and different modes and assigned different priorities. This is shown in the figure 4.1 below.

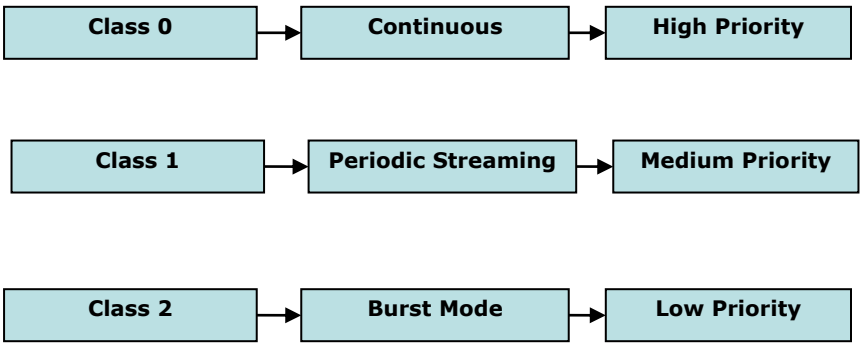



**Figure 4.1: Priority classes with respect to modes**

The procedure for burst preemptive dropping is shown in the figure 4.2 below. Consider there is burst of MP or LP class which is already scheduled. After that a request for HP class burst comes for same destination port. Scheduler tries to schedule burst but does not find any free wavelength, then scheduler tries to find overlapping LP burst and if it finds then it drops. If it does not find any LP burst then it tries to find overlapping MP class burst and if it finds then it drops and schedule HP class burst. If scheduler does not find any LP or MP class burst, then incoming HP class is dropped.

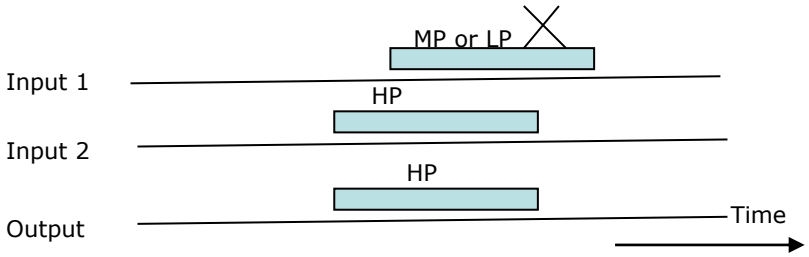



**Figure 4.2: Preemptive dropping at core node**

Extra offset time for HP class burst is increased four times of the LP class burst. No extra offset time is assigned to MP class.

# 5. Performance Evaluation

In order to evaluate performance of proposed technique, simulation technique is used. First we have simulated OBS network, then we have simulated FTM under similar parameters and its performance is compared with OBS, and then we have simulated FTM with QoS provisioning and its performance is compared with both OBS and FTM. Simulation is written in java language.

## 5.1 Network Model

The network model which is used for simulation is shown in the figure. There are three source edge nodes (ingress nodes) which generate bursts for single destination edge node (egress node) which are connected via single core node. In case OBS network, traffic is not distinguished but in case of FTM traffic is distinguished as follows:

- Source A generates Burst Mode traffic
- Source B generates Periodic Mode traffic
- Source C generates Continuous Streaming Mode traffic

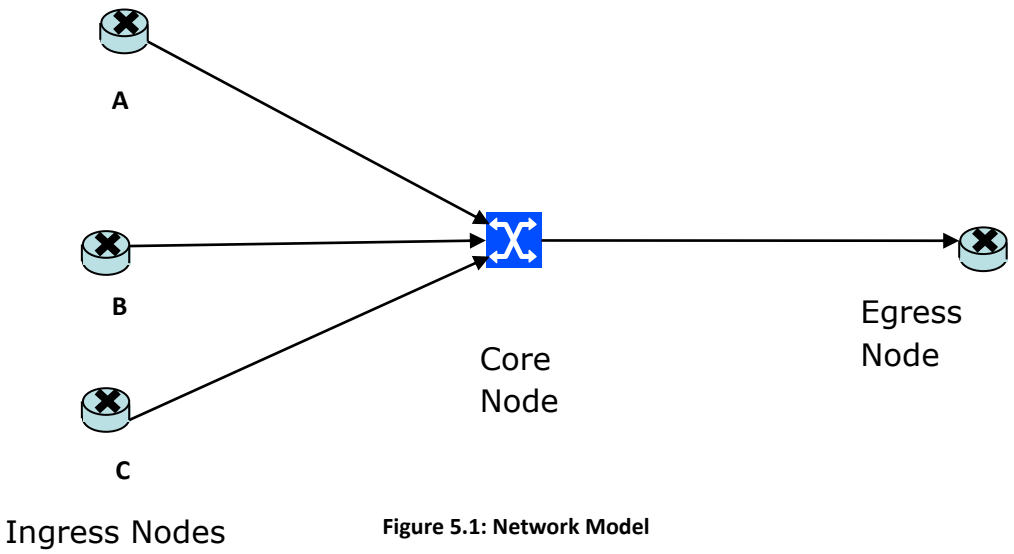



**Figure 5.1: Network Model**

## 5.2 Simulation Technique

We have used discrete event simulation technique. For traffic generation we have used Poisson arrival of bursts at core node. Detailed description of Poisson process is given in Appendix A. Mean burst size is assumed as 50 kilo bytes. Burst size is negative exponentially distributed and is calculated in the following formula:

- Burst size= - (mean burst size) * log(r)

where r is the random number between 0 and 1. Exponential distribution is described briefly in Appendix B. Data rate is kept at 10 Giga bits per second. Propagation delay is considered 1 mili second. Arrival rate $\lambda$ is taken from 0.01 to 0.9 for different readings. Burst inter-arrival time is also negatively exponentially distributed and is calculated using following formula:

- Inter arrival time = - $1/\lambda$ * log (r)

For burst signaling, JET protocol is used and for wavelength reservation LAUC-VF algorithm is used. Simulation parameters were considered as in [23].

## 5.3 Performance Metrics

Performance metrics which are used to evaluate performance of FTM are burst loss ratio, link utilization and normalized throughput against load offered to the network. All of the metrics are calculated using following formulae:

- **Load** = mean burst length / mean inter-arrival time
- **Total Load in FTM** = Load in OBS Mode + Load in Periodic Mode + Load in Continuous Mode
- **Burst Loss Ratio** = No of bursts dropped / Total number of bursts send
- **Network Throughput** = Total no of bits passed / Total Simulation Time.
    - Network throughput is measured in Gbits/sec
- **Bandwidth Utilization** = Average Network Throughput / Available Bandwidth
- **Normalized Throughput** = Total no of burst scheduled / Total number of bursts send

- **95 % Confidence Interval** = mean ± (1.96 * standard deviation)

## 5.5 Implementation Detail

Implementation phase of our thesis is divided into three phases i.e.

- OBS Implementation
- FTM Implementation
- FTM Implementation with QoS Provisioning

### 5.5.1 OBS Implementation Detail

The first phase in implementation detail is OBS implementation. First phase in OBS implementation is traffic generation which is done using Poisson distribution. Traffic is generated and placed in event list as shown in the figure 5.2 below. Clock is incremented till simulation time. Source is selected randomly from 3 different sources. Then inter-arrival time and burst length is calculated using exponential distribution. Then control packet is generated and placed in event list which contains all of the information of incoming burst. In this way event list contains all the control packets which will be scheduled using scheduling algorithm.

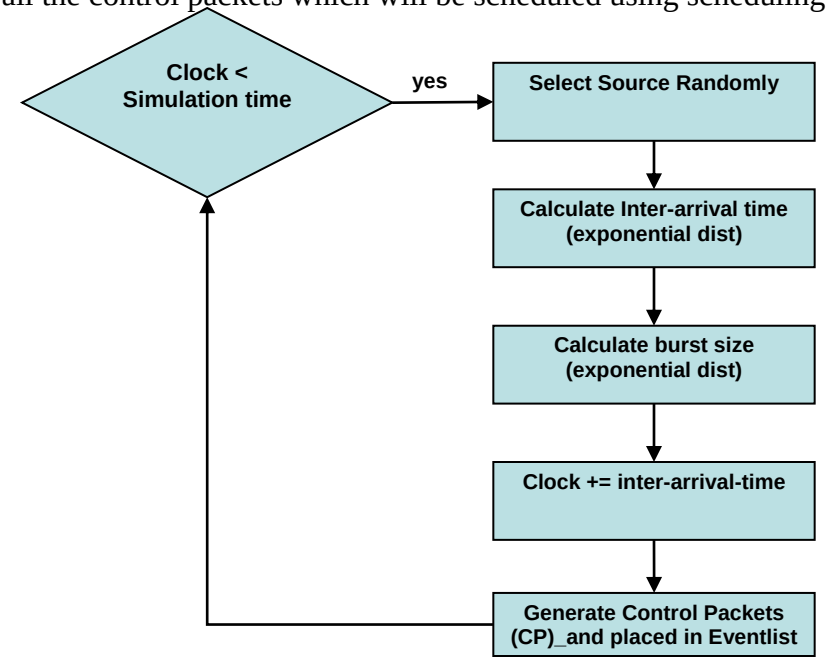



**Figure 5.2: Traffic Generation in OBS**

Next step is shown in the figure 5.3. In this step eventlist is read and channel is schedule using LAUC-VF algorithm. No of burst send and dropped will be incremented depending upon successful or failure of burst reservation.

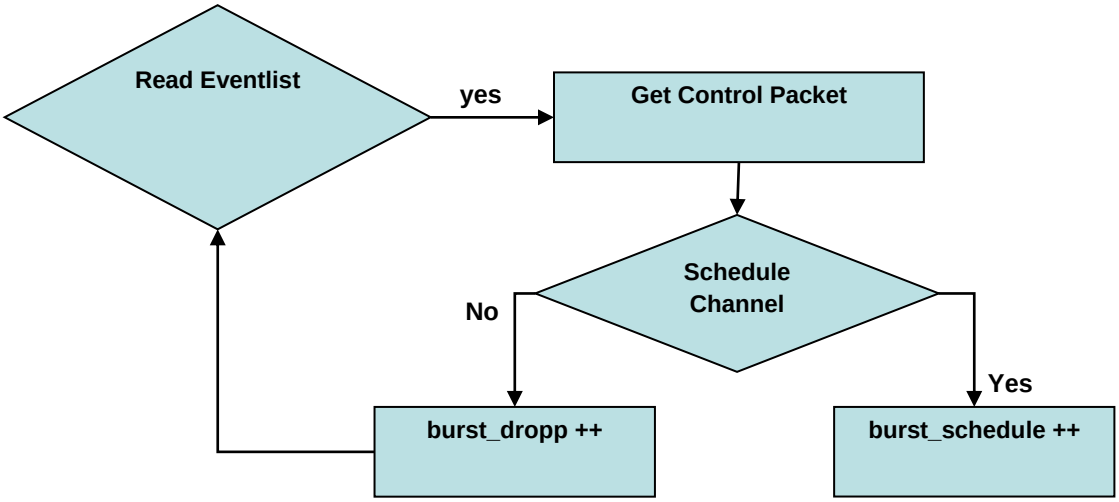



**Figure 5.3: Read Eventlist & Schedule Channel**

Detail of implementation of LAUC-VF algorithm is described in figure 5.4, 5.5 and 5.6. LAUC-VF first tries to schedule channel using void filling as shown in the figure 5.4

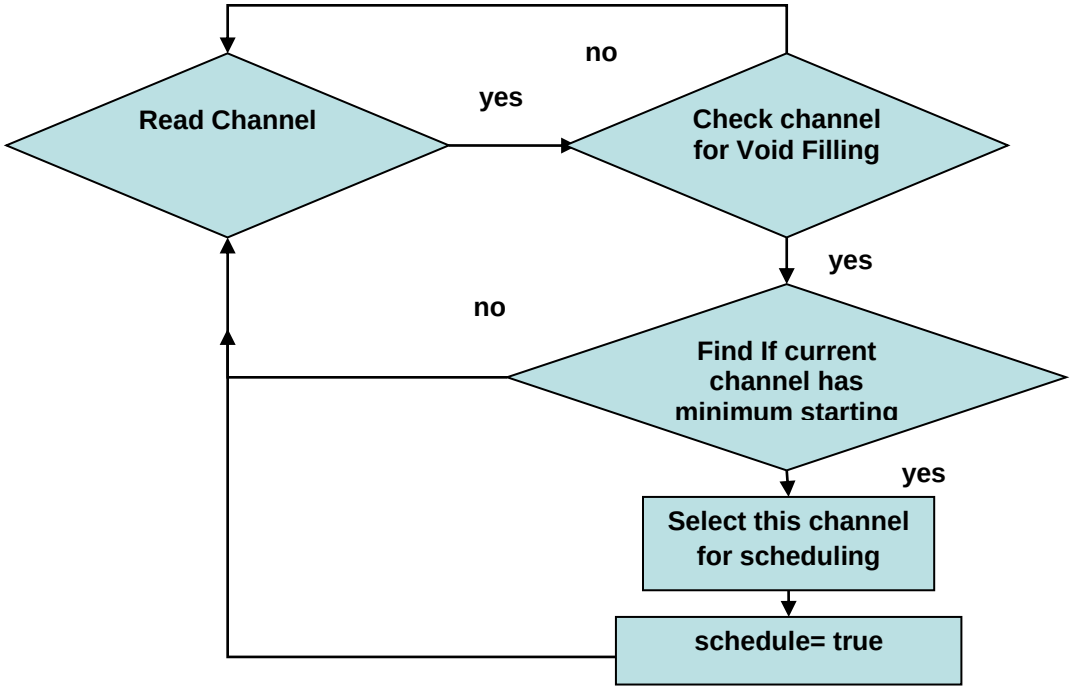



**Figure 5.4: LAUC-VF**

If channel is not successful schedule then LAUC algorithm is used which is shown in the figure 5.5 and 5.6.

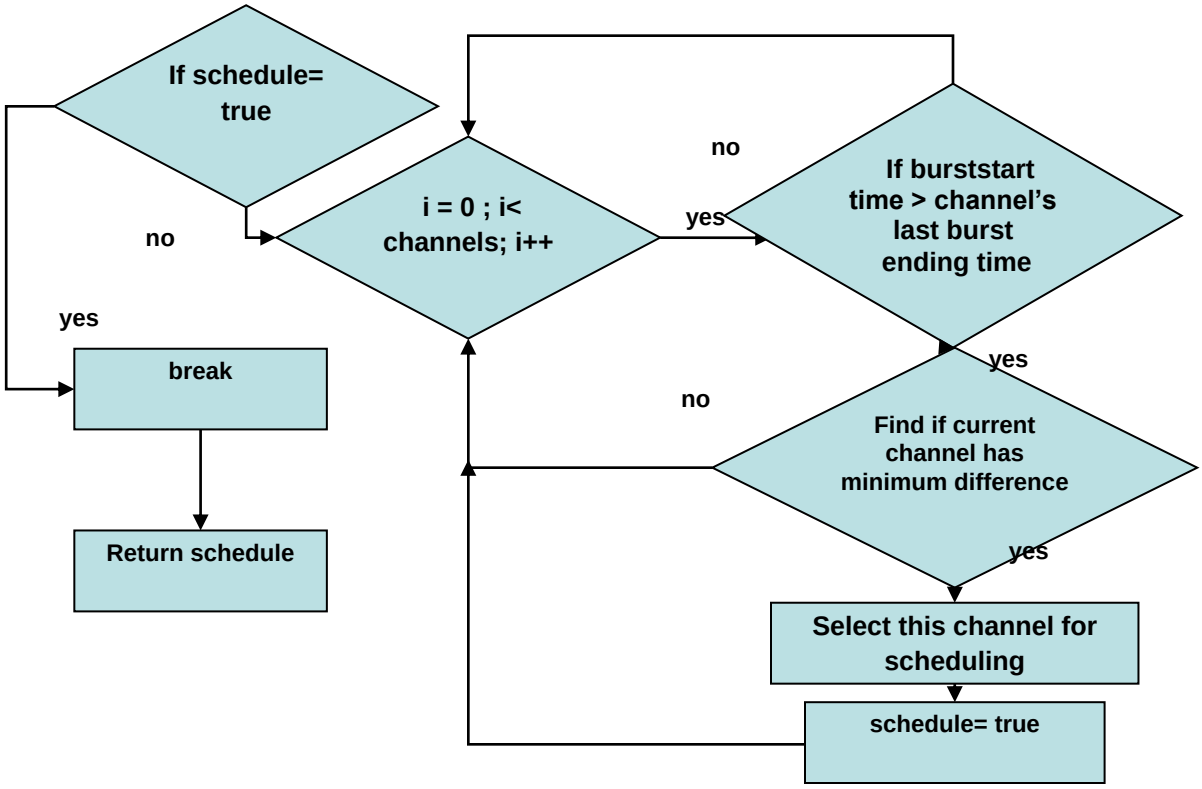



Figure 5.5: Schedule Channel using LAUC

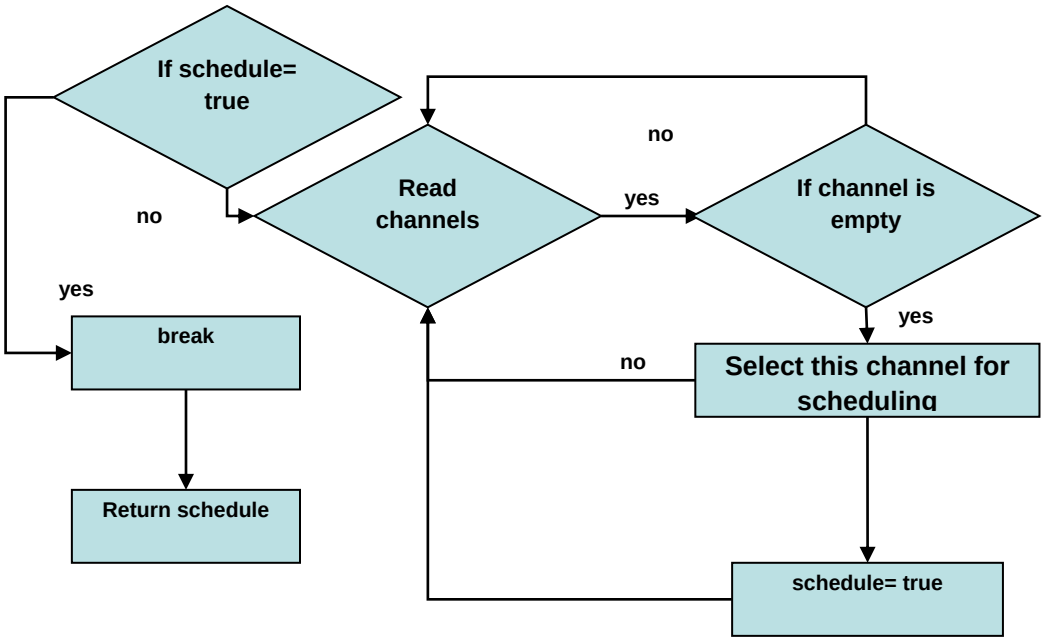



Figure 5.6: Schedule Channel using LAUC (Empty)

### 5.5.2 FTM Implementation Detail

First step in FTM implementation detail is also traffic generation. Here we have three types of traffic for 3 different modes of FTM. Control packets for each type of traffic is generated and placed in the event list. In the end event list is sorted. Burst mode traffic generation is similar to traffic generation in OBS and is shown in the figure 5.7. Only priority assignment step is additional traffic generation of burst mode in FTM.

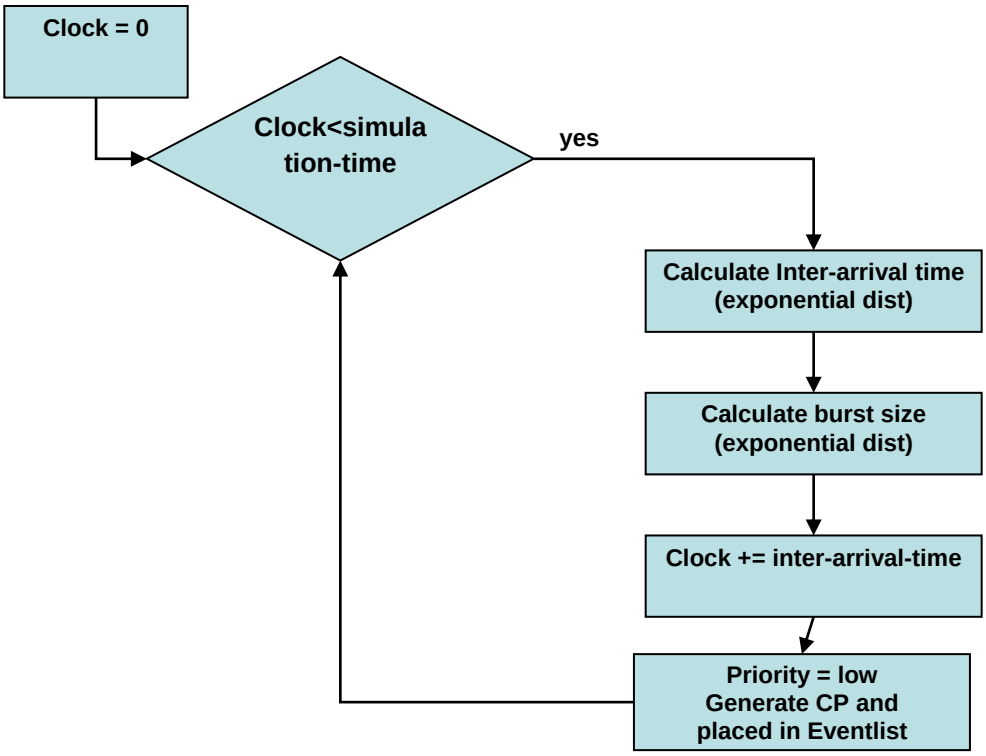



**Figure 5.7: Traffic generation of burst mode**

Figure 5.8 shows traffic generation of continuous mode in FTM. Burst size and inter-arrival time is calculated using exponential distribution. Extra offset time is increased by 4 times and high priority is assigned to continuous mode traffic. We have considered 2 streams for Periodic Streaming modes of FTM. i.e. 1 Mb stream and 0.5 Mb stream. In 1 Mb stream, there will be 20 bursts while in case of 0.5 Mb stream, there will 10 burst in stream considering the mean burst size of 50 bytes. Figure 5.9 shows traffic generation of periodic streaming mode. Gap between successive burst is kept equal to inter-arrival time of first burst in periodic streaming mode.

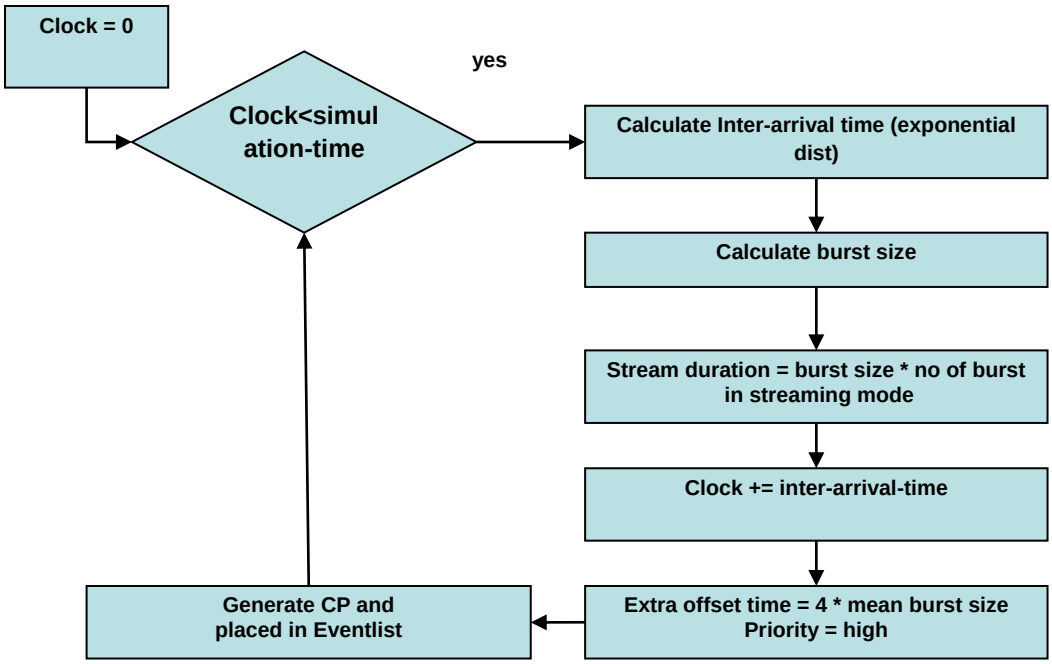



**Figure 5.8: Traffic Generation of Continuous Mode**

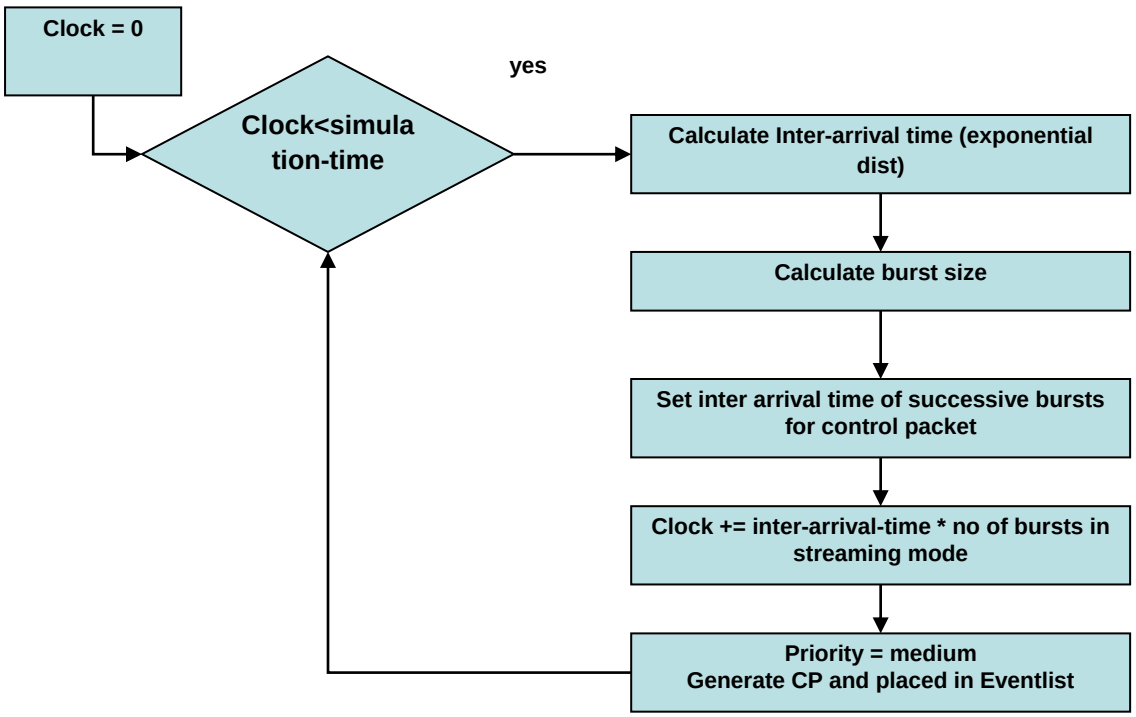



**Figure 5.9: Traffic Generation of Periodic Streaming Mode**

For scheduling, same LAUC-VF algorithm is used. Scheduling technique is shown in the figure 5.10. Burst mode and continuous mode scheduling is same as in OBS. Only difference is of length of streams in continuous mode which will be 10 bursts in case of half Mb stream and 20 bursts in case of 1 Mb stream. In this way whole stream is schedule using LAUC-VF in one

channel. For periodic streaming, we have considered 2 cases. If we use LAUC for periodic stream then whole stream will be schedule in one channel but if we use LAUC-VF algorithm, then different bursts in stream will be scheduled on different channels. We have evaluated performance of both cases and is described in results section of this thesis.

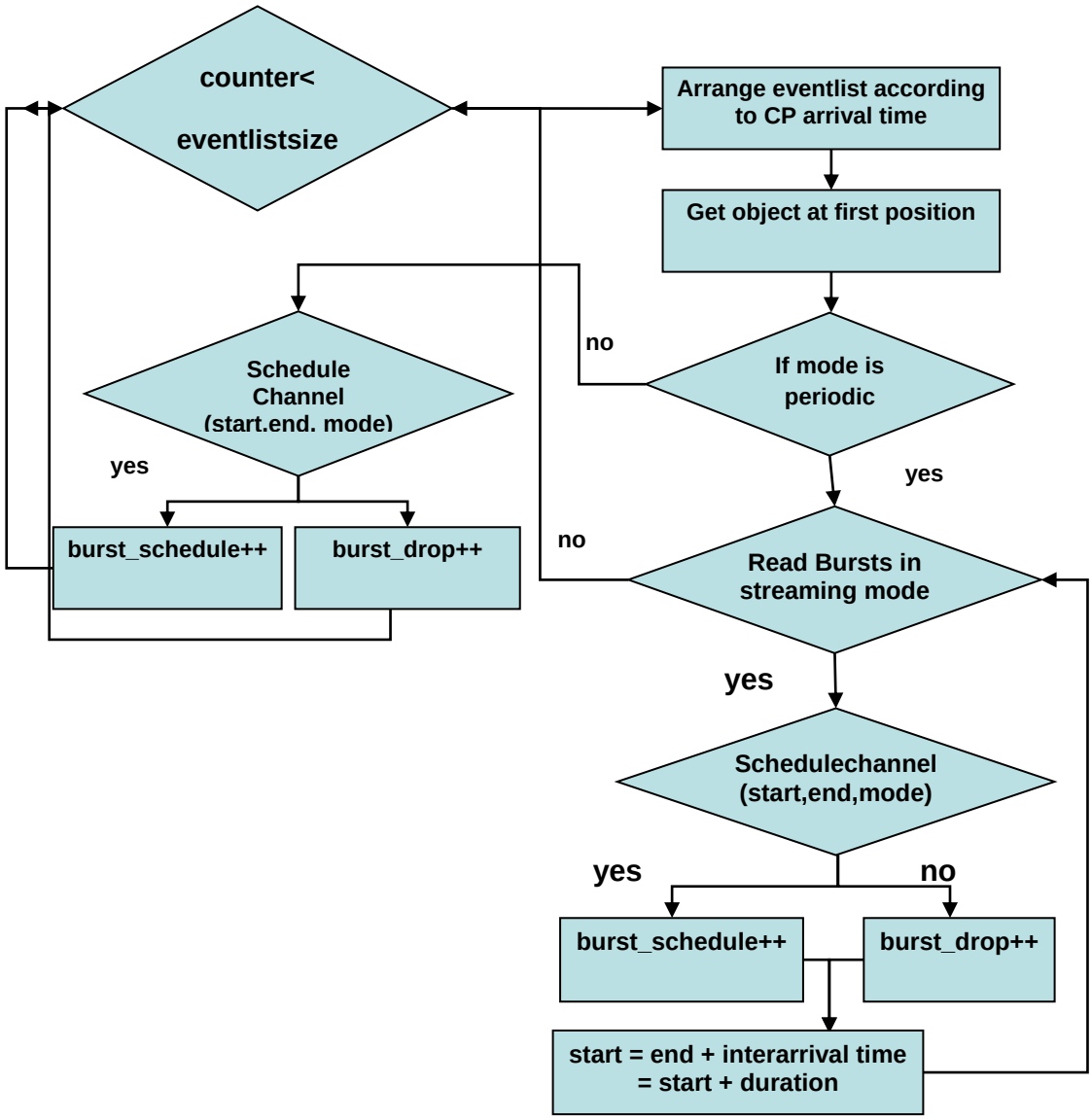



**Figure 5.10: Scheduling in FTM**

### 5.5.3 FTM Implementation with QoS Provisioning

Figure 5.11 shows QoS provisioning in FTM. High priority is assigned to continuous mode traffic. If high priority stream is not successfully scheduled using LAUC-VF algorithm, then pre-emptive dropping mechanism is used which is shown in the figure 5.11. First of all, overlapping bursts of burst mode is investigated. If overlapping bursts of burst mode is found then minimum of these burst is dropped and continuous stream is scheduled. If bursts of burst mode did not find then algorithm tries to find overlapping periodic mode burst. If overlapping bursts of periodic mode find then minimum number of these bursts dropped and continuous stream is scheduled. If both cases fail, then continuous stream is dropped.

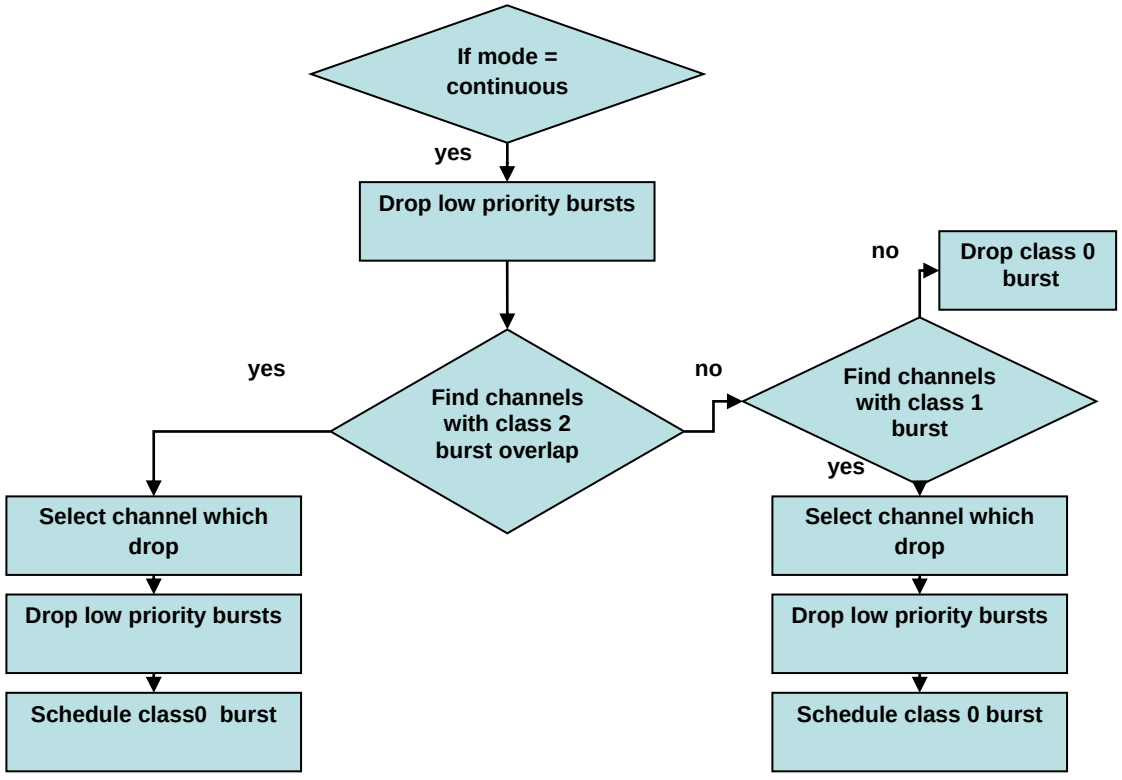



**Figure 5.11: QoS Provisioning in FTM**

## 5.6 Results

In this section, we have presented detailed description of different results which are generated by using different combinations of parameters. We have considered following cases for results and have done performance evaluation of OBS, FTM and FTM with QoS provisioning in each case. Our results also show 95 % confidence interval. Following cases are considered for simulation:

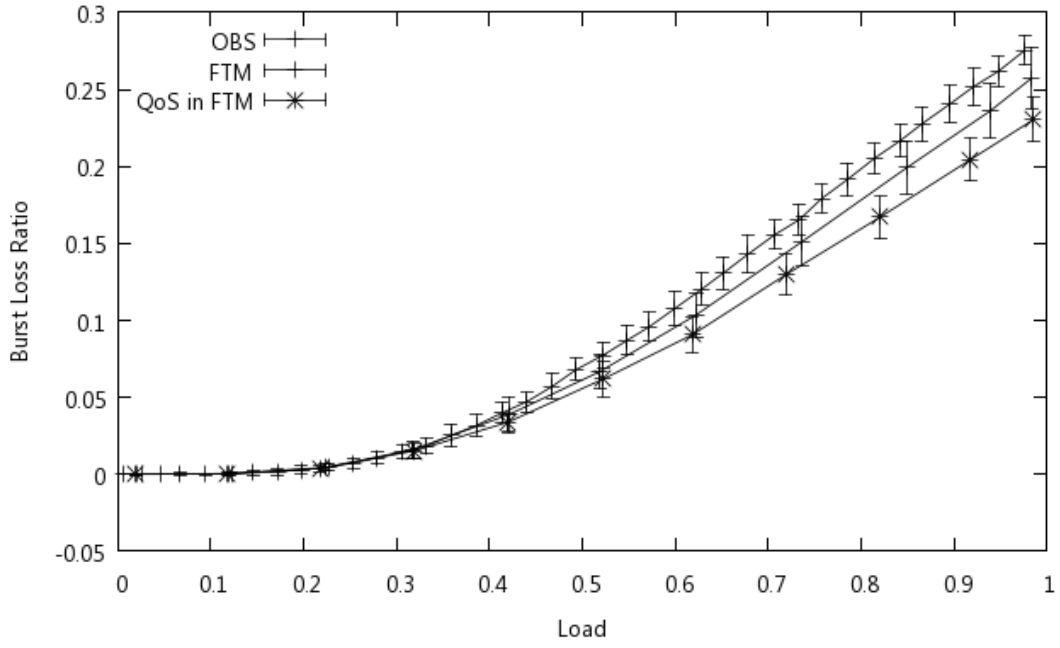
1. 1Mb Streams with equal load, 6 wavelengths and periodic streaming with LAUC-VF.
2. 0.5 Mb Streams with equal load, 6 wavelengths and periodic streaming with LAUC-VF.
3. 1Mb Streams with different load, 6 wavelengths and periodic streaming with LAUC-VF.
4. 1Mb Streams with equal load, 6 wavelengths and periodic streaming with LAUC.
5. 1Mb Streams with equal load, 12 wavelengths and periodic streaming with LAUC-VF.
6. 1Mb Streams with equal load, 18 wavelengths and periodic streaming with LAUC-VF.

In first case, 1 Mb streams of both continuous and periodic mode is considered. Load is equally divided among all three modes, 6 wavelengths for data channels are considered, and for scheduling of periodic streaming LAUC-CF algorithm is used. When we use LAUC-VF algorithm then all the bursts in periodic stream will not schedule in same channel i.e. some bursts might take different channels. When we use LAUC algorithm for periodic streaming then all the bursts in the stream will schedule in the same channel. We have evaluated performance of periodic streaming mode by using both algorithms. In case of periodic streaming with LAUC-VF algorithm, control packet will have to store additional information for each burst while in case of LAUC only a gap between successive bursts is stored.

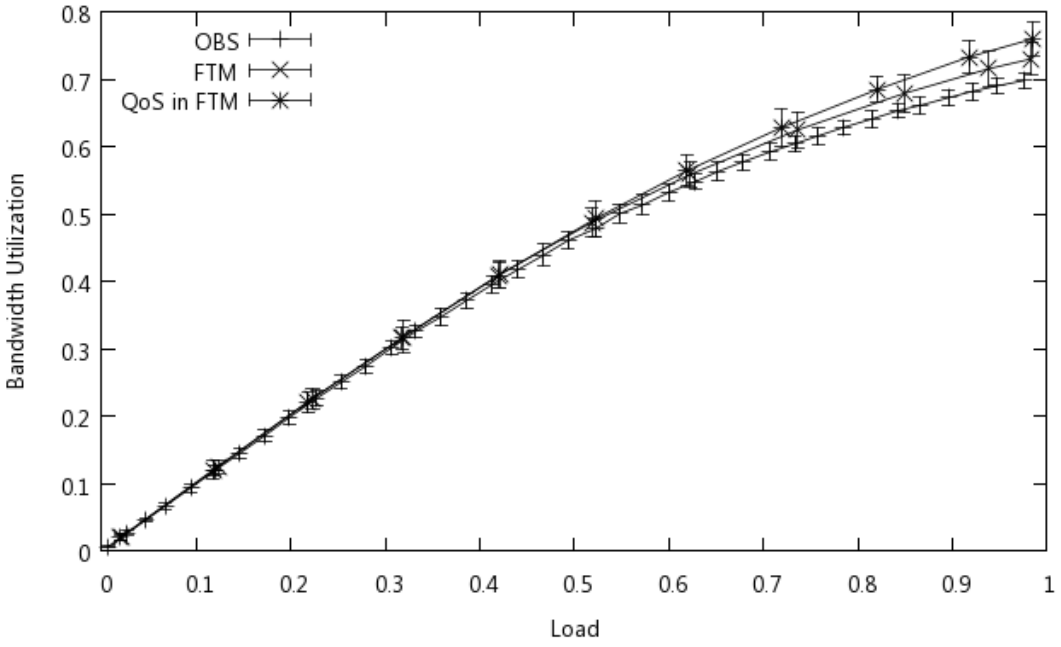
Continuous stream will follow same channel in all cases because we have used LAUC-VF for whole stream and not for individual burst. We also tried different lengths of streams and also tried different loads for different modes. Different performance metrics are calculated against load. Load is kept from 0 to 1.

Different results are presented from figure 5.12 to 5.17. Each figure is subdivided into 4 parts for different performance metrics calculation.

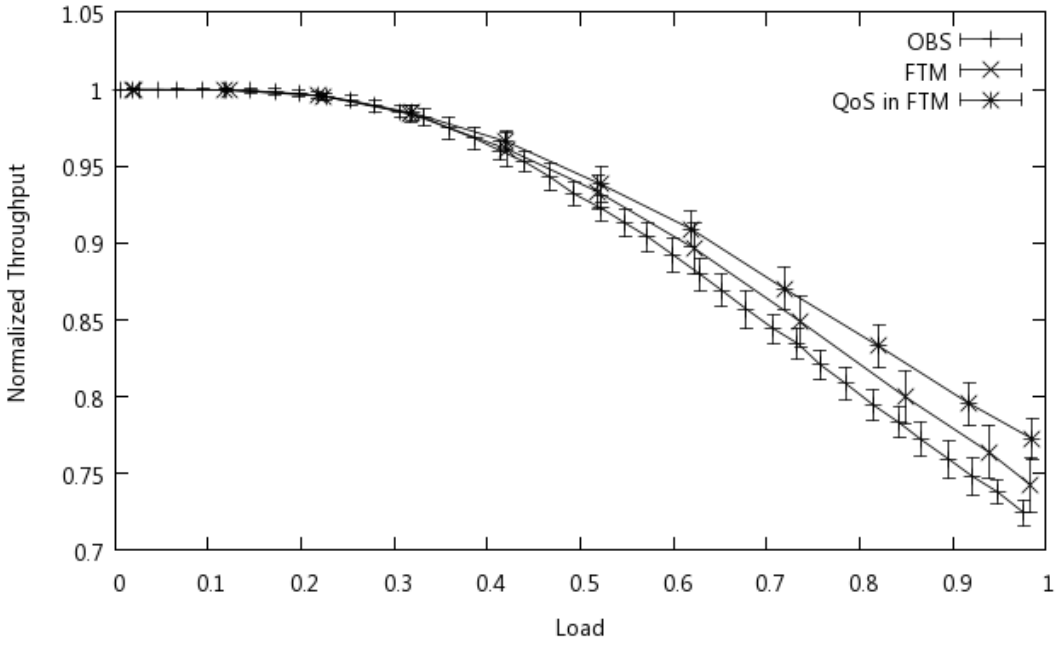
Consider figure 5.12. Different performance metrics i.e. burst loss ratio, bandwidth utilization, normalized throughput and burst losses in different classes are calculated against load. In this case 1Mb streams with equal load for different modes by using 6 wavelengths. For periodic streaming LAUC-VF algorithm is used so that bursts in periodic streaming will follow different channels. Figure 5.12 (a) shows that FTM has low burst losses as compared to OBS and this ratio is further reduced by using our proposed QoS mechanism in FTM. Figure 5.12 (b) shows that our proposed mechanism has higher bandwidth utilization as compared to both FTM and OBS. Same is the case with normalized throughput as well. In figure 5.12 (d), burst loss ratio of 3 classes are compared with and without QoS provisioning which shows that by employing QoS mechanisms continuous stream mode has low burst losses as compared to other classes due to high priority is assigned to this mode.



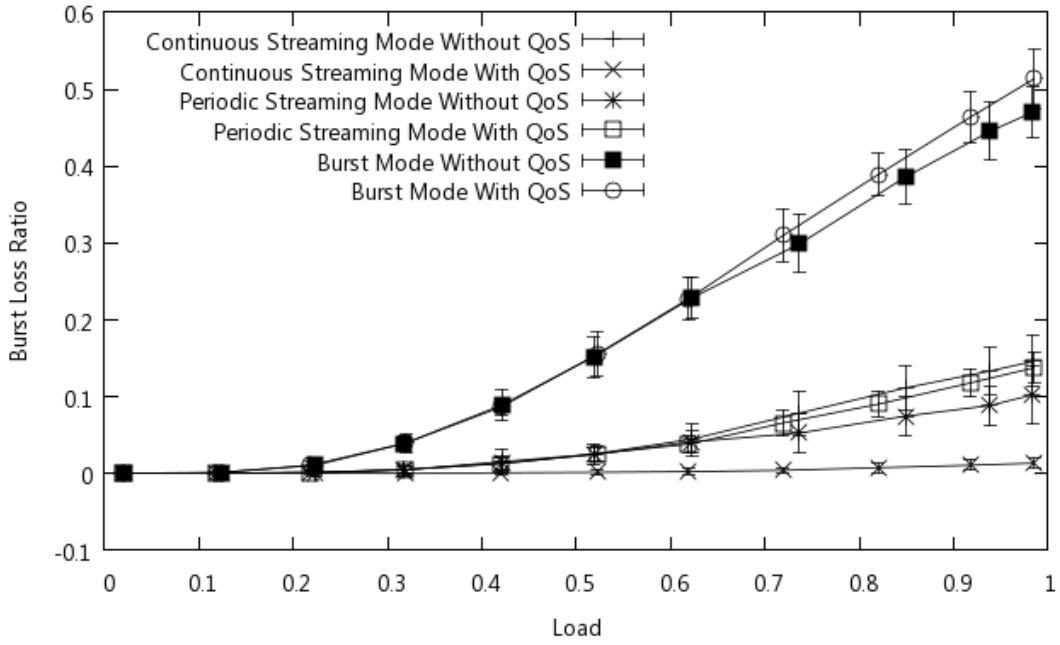

(a) Load vs Burst Loss Ratio




(b) Load vs Bandwidth Utilization




(c) Load vs Normalized Throughput

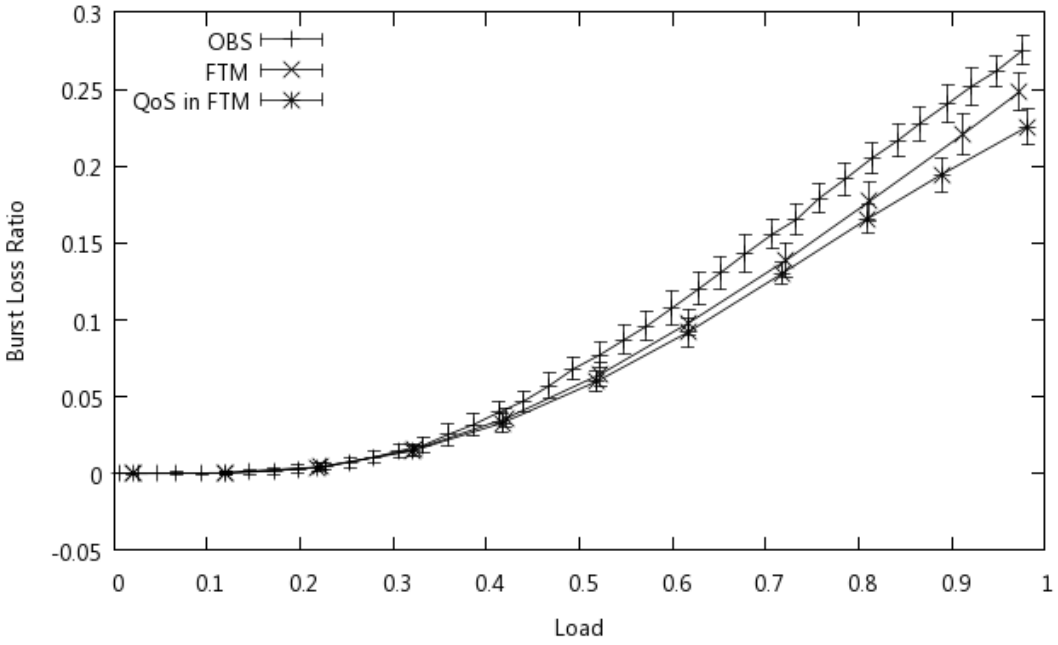



(d) FTM Modes with & without QoS Provisioning

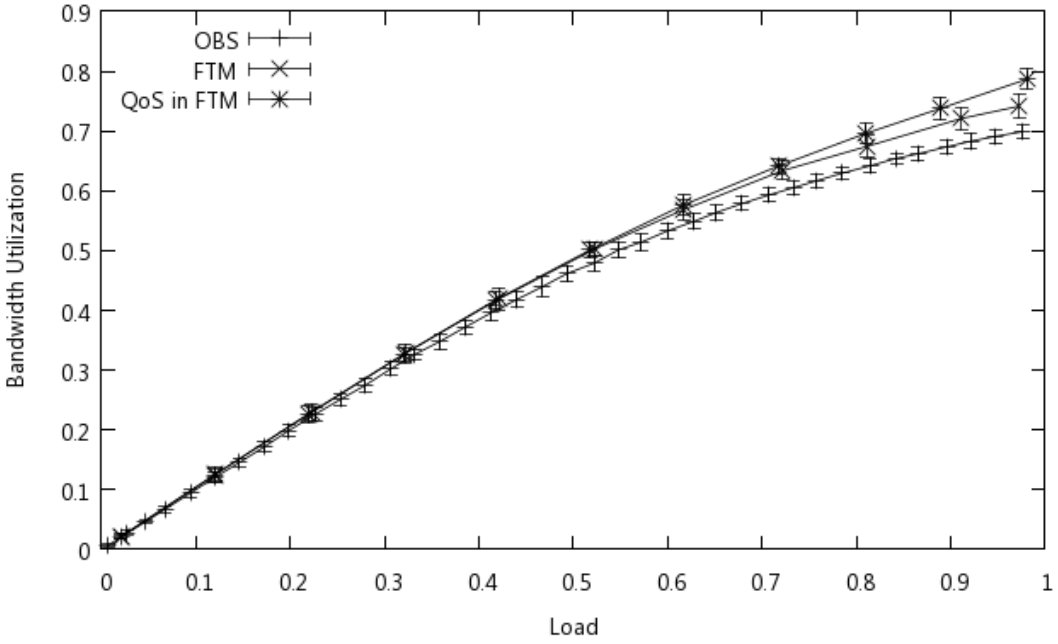
**Figure 5.12 : Load vs. burst loss ratio, bandwidth utilization, normalized throughput and burst losses in different classes which are calculated by using 1Mb streams with equal load, 6 wavelengths & periodic streaming with LAUC-VF.**

Performance is improved due to streaming modes of FTM and specially continuous streaming mode of FTM, because by using streaming modes maximum number of bursts are scheduled which ultimately results in lower burst losses and higher bandwidth utilization and normalized throughput.

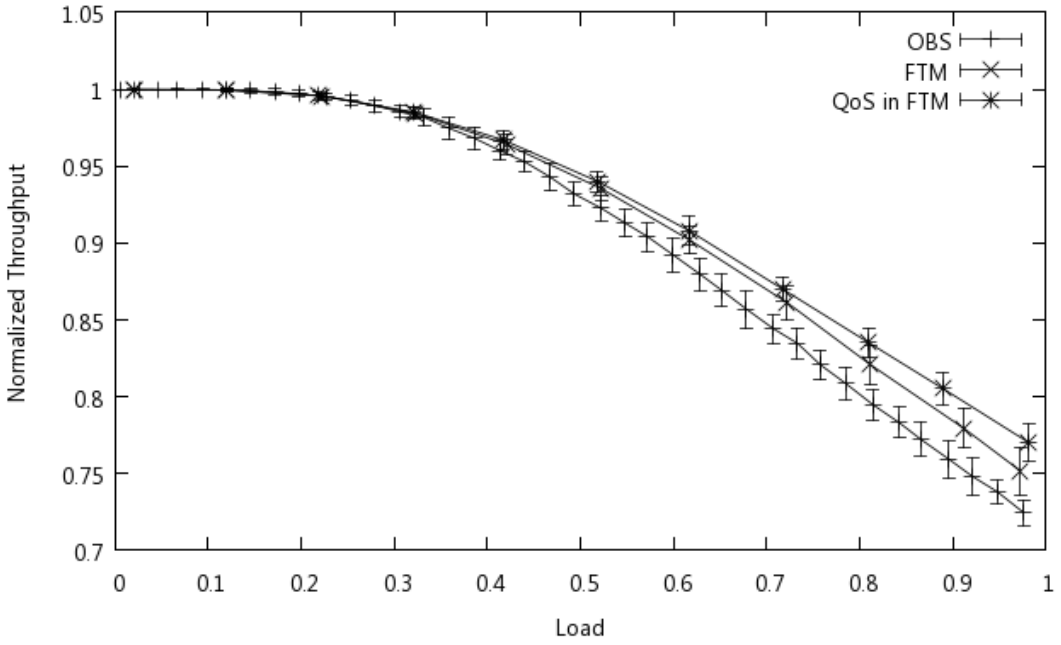
Same case is considered again by taking 0.5 Mb streams and results are shown in the figure 5.13. There is not much difference between these results with the results in figure 5.12. However performance of proposed QoS mechanism is improved in both cases as compared to OBS and FTM.



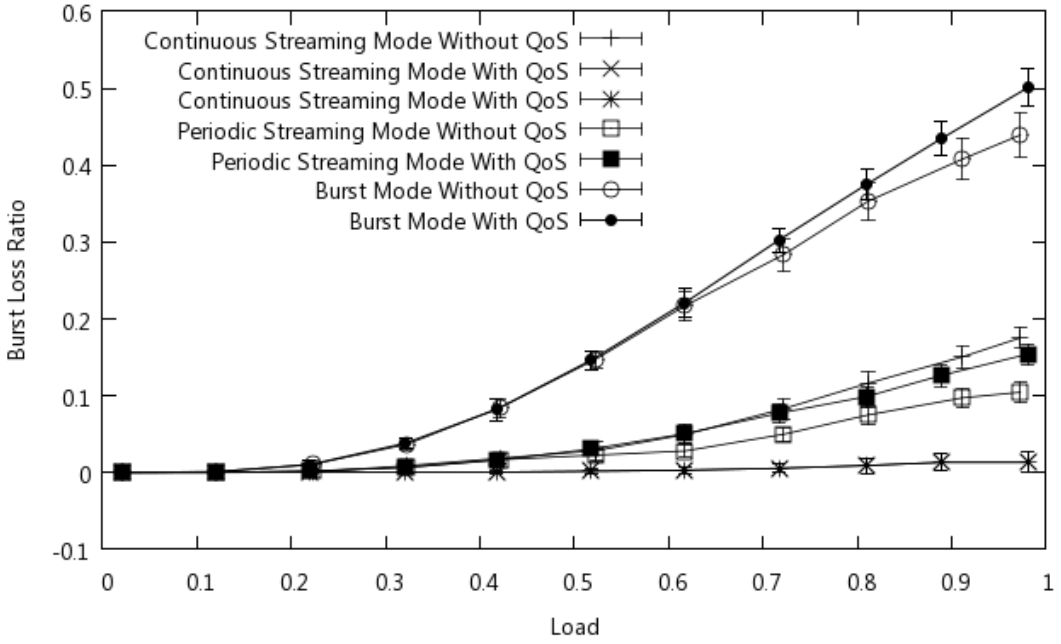

(a) Load vs Burst Loss Ratio




(b) Load vs Bandwidth Utilization




(c) Load vs Normalized Throughput

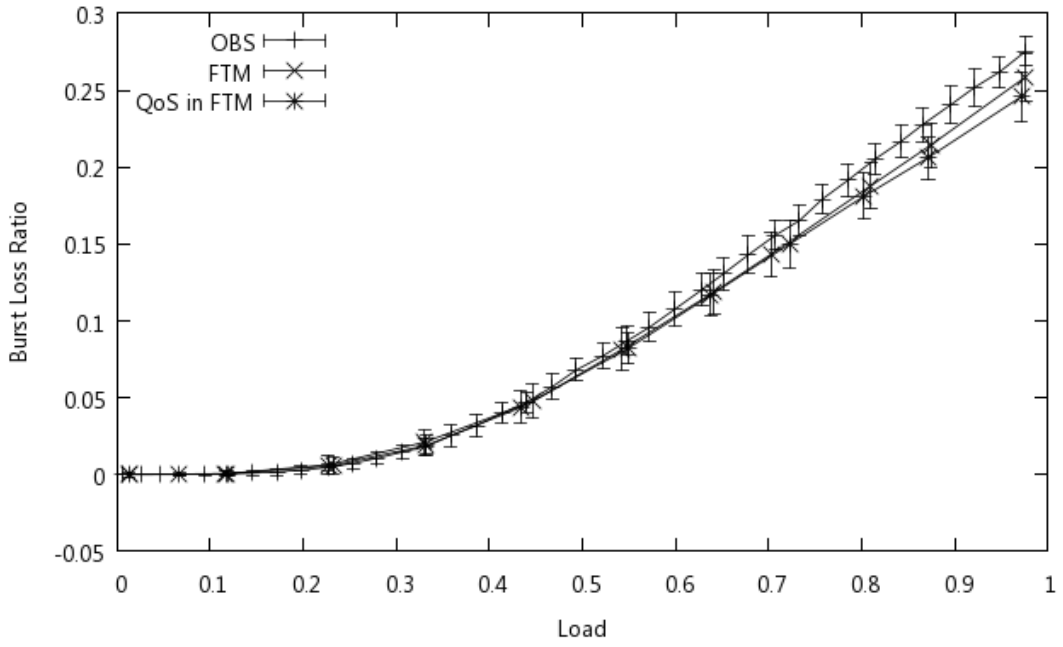



(d) FTM Modes with & without QoS Provisioning

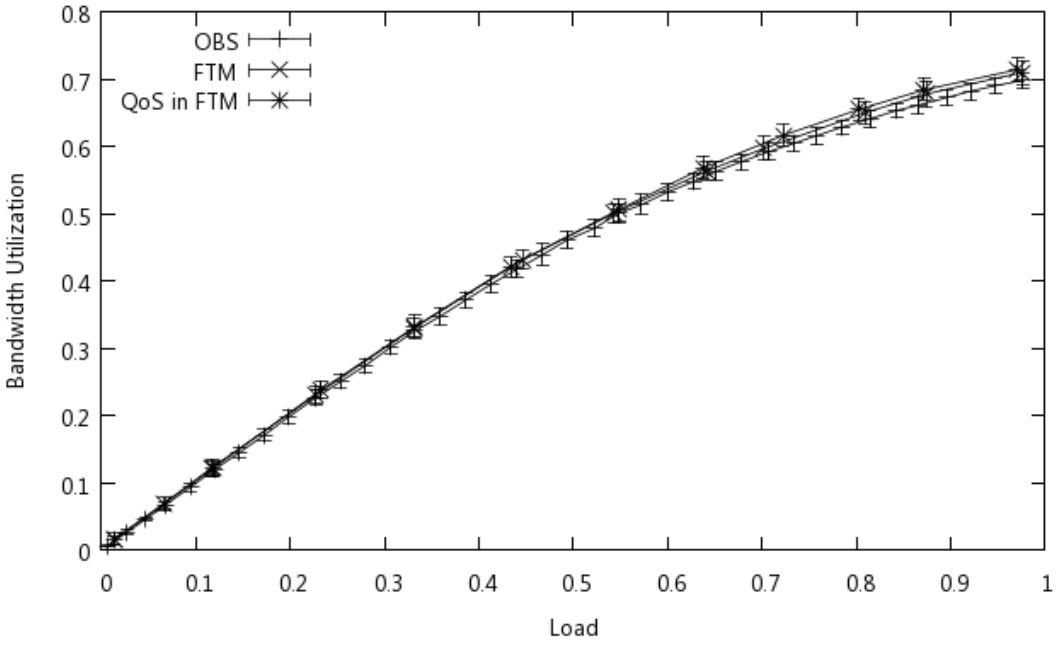
**Figure 5.13: Load vs. burst loss ratio, bandwidth utilization, normalized throughput and burst losses in different classes which are calculated by using 0.5 Mb streams with equal load, 6 wavelengths & periodic streaming with LAUC-VF.**

Results with different loads are generated in figure 5.14. In this case, load for burst mode is kept at 50 %, load for periodic mode is kept at 35 % and load for continuous mode is kept at 15 %.

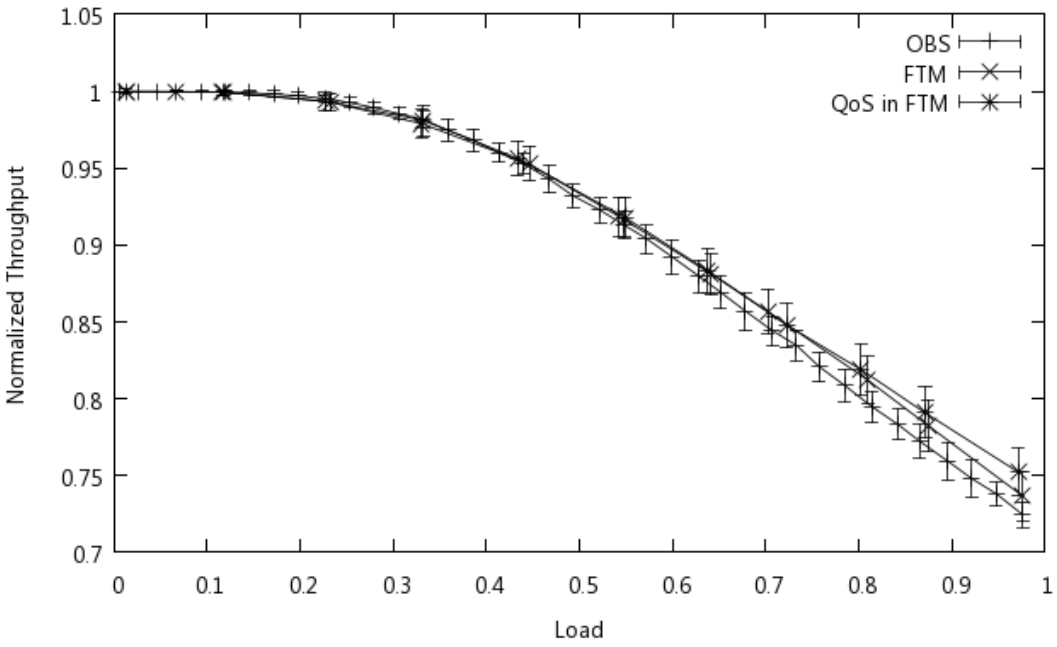
Figure 5.14 shows that there is still improvement of performance in all the metrics but not very much as was in previous two cases. The reason for this is that load for continuous mode is decreased from 33% to 15 % in this case and also overall load of streaming modes i.e. 66% is decreased to 50 % in this case. As streaming mode traffic is decreased, then FTM will behave like OBS and when load of streaming modes is increased then there will be more improvement in performance of overall network.



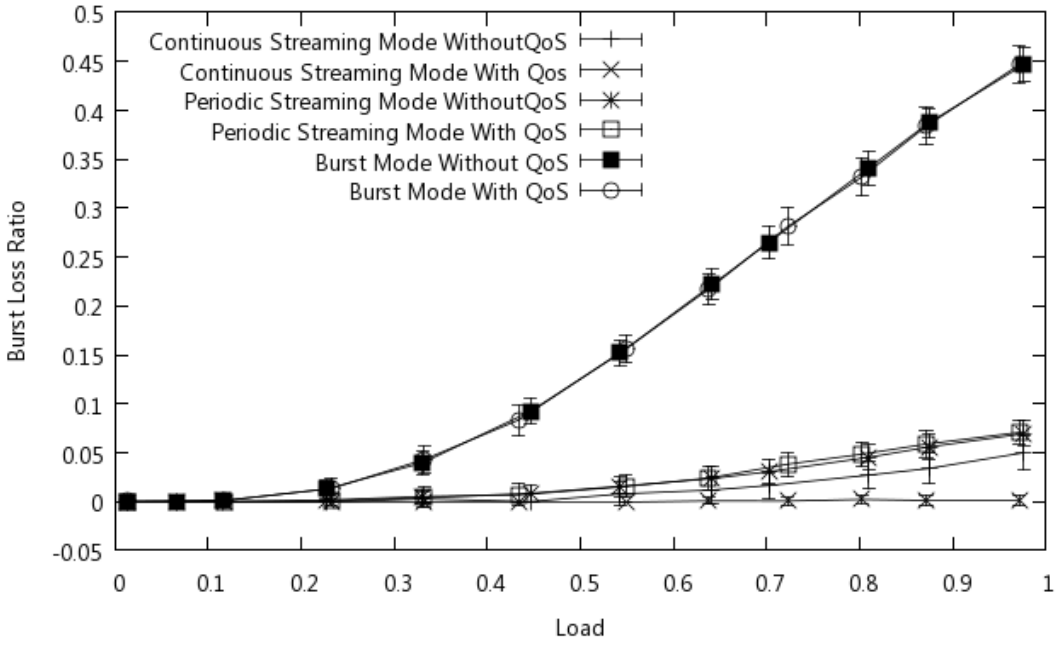

(a) Load vs Burst Loss Ratio




(b) Load vs Bandwidth Utilization



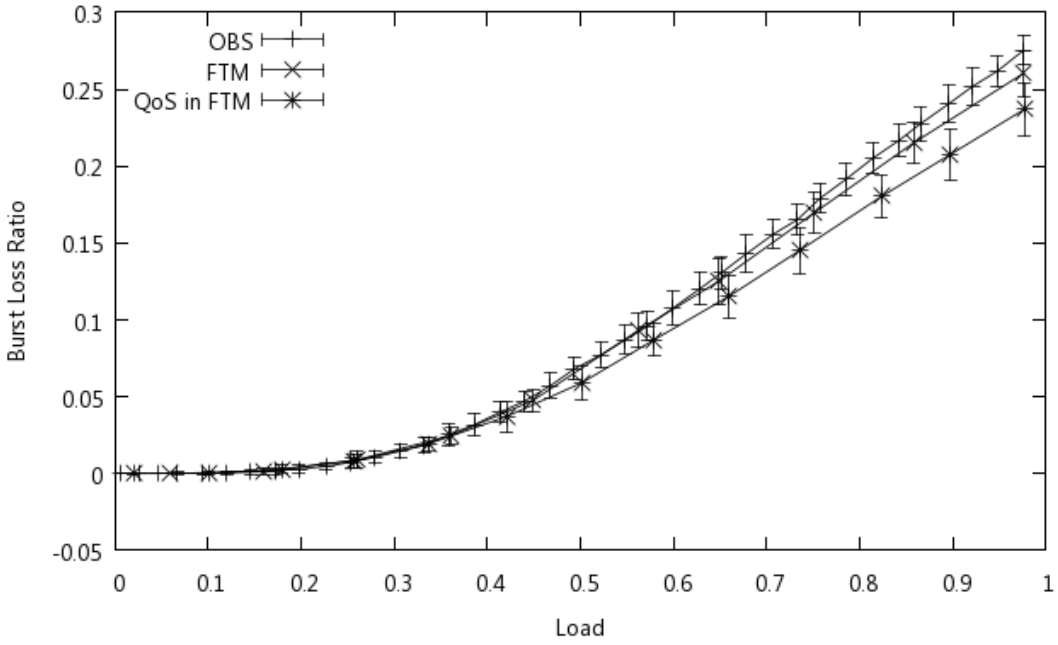

(c) Load vs Normalized Throughput



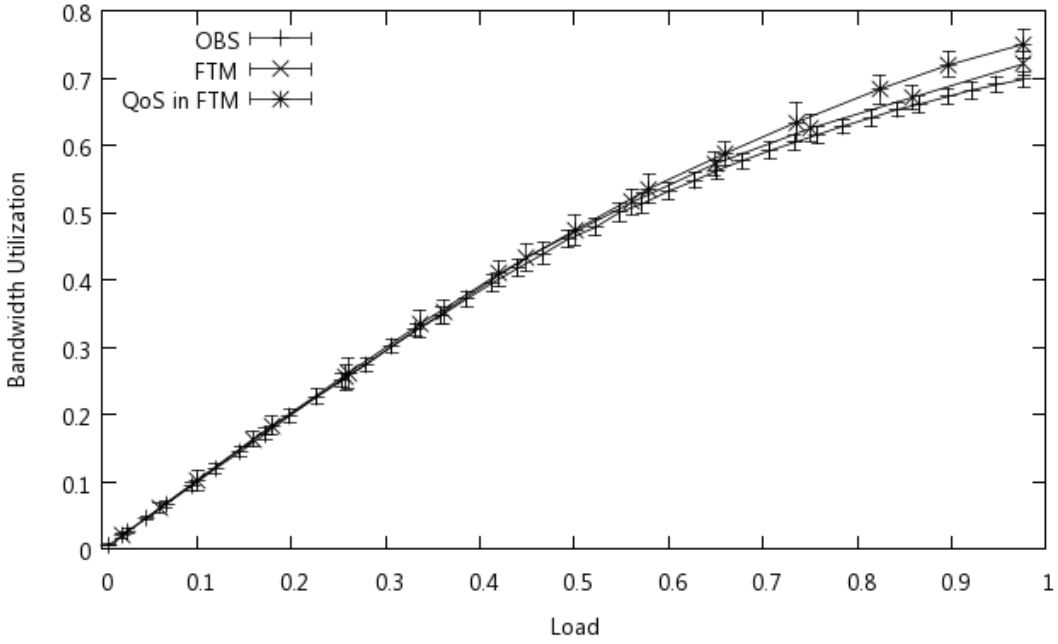

(d) FTM Modes with & without QoS Provisioning

**Figure 5.14: Load vs. burst loss ratio, bandwidth utilization, normalized throughput and burst losses in different classes which are calculated by using 1 Mb streams with different load, 6 wavelengths & periodic streaming with LAUC-VF.**

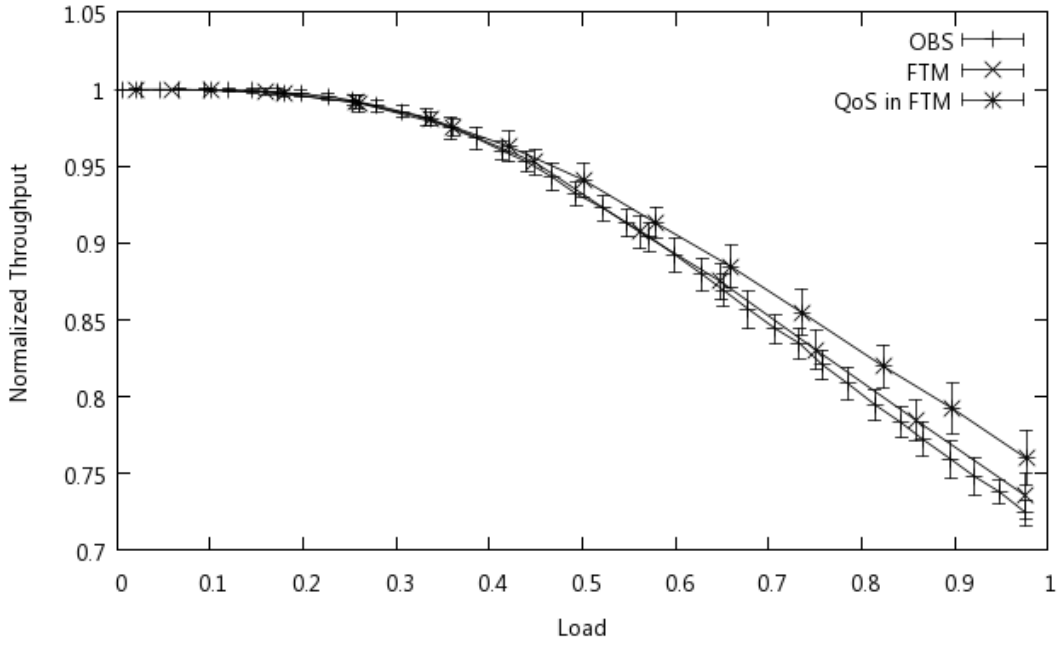
In Figure 5.15, another algorithm i.e. LAUC is used for periodic streaming mode. In this way all the burst in periodic streaming will be tried to schedule in same channel. Results show that overall network performance is improved with QoS provisioning in FTM, but performance of FTM has been declined as compared to previous cases but still it is better than OBS. Decline to some extent is caused by LAUC algorithm which has high burst losses and less bandwidth utilization as compared to LAUC-VF. This is shown in the figure 5.15 (d), as it is clearly seen that periodic mode has slightly high burst losses with and without QoS provisioning in FTM as compared to previous cases.



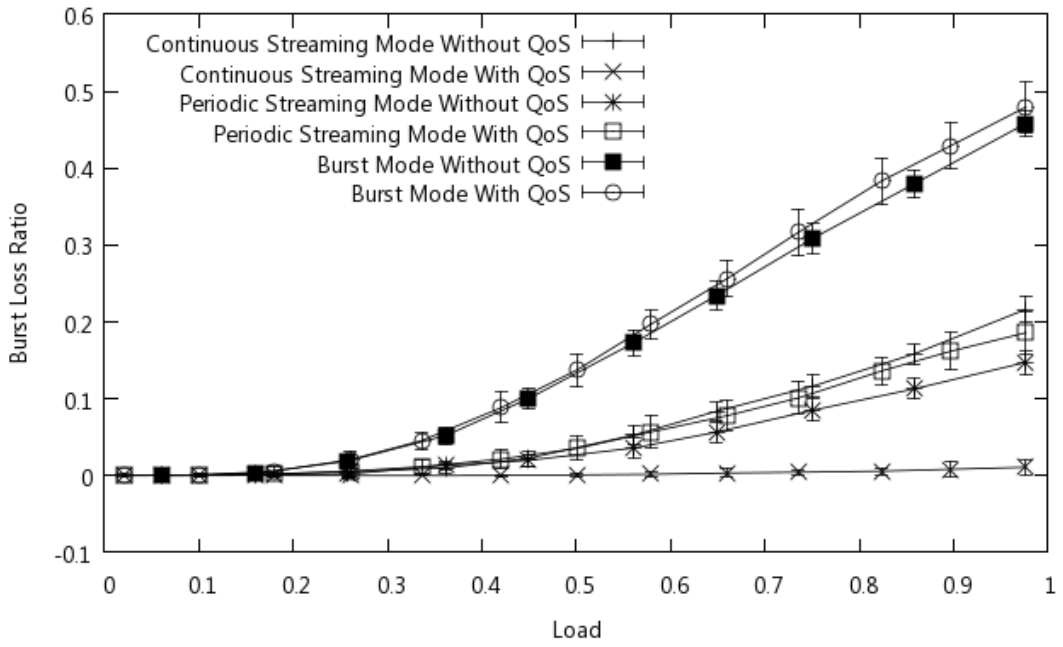

(a) Load vs Burst Loss Ratio




(b) Load vs Bandwidth Utilization




(c) Load vs Normalized Throughput

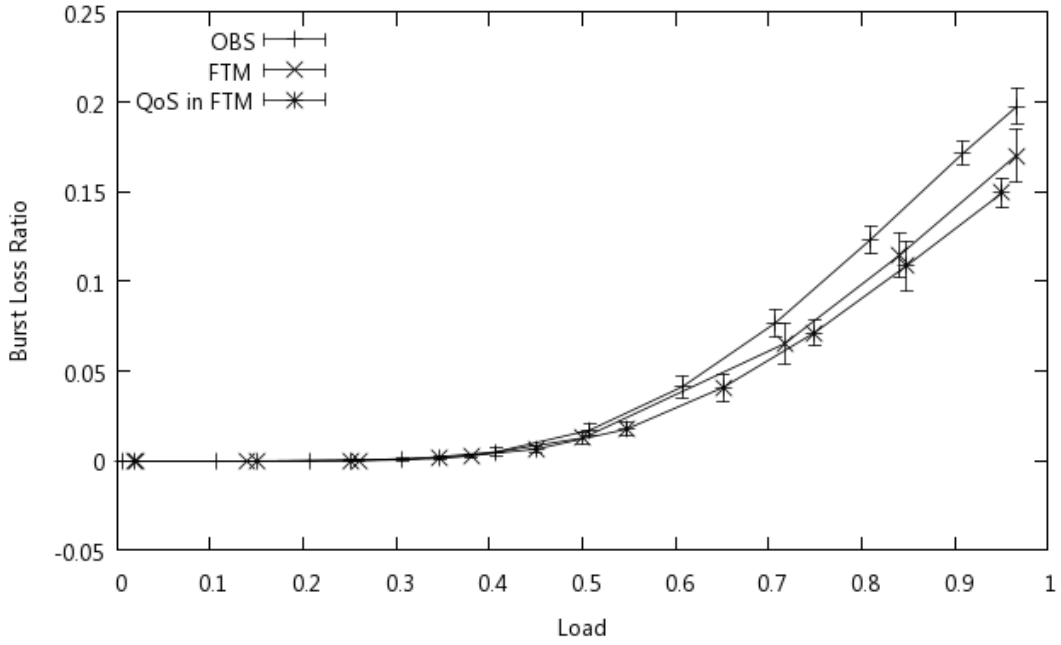



(d) FTM Modes with & without QoS Provisioning

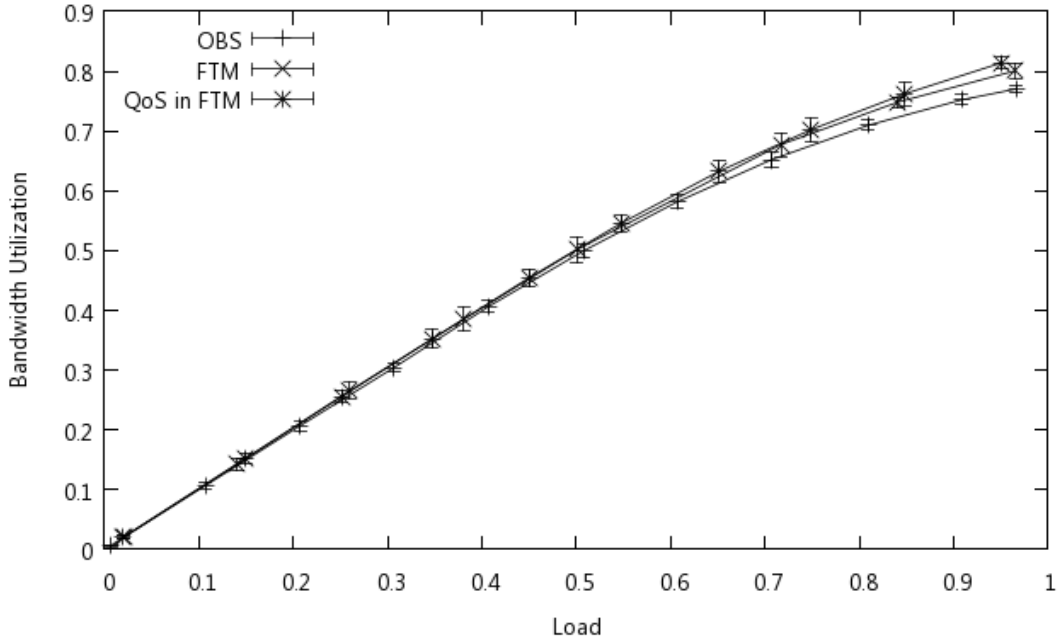
**Figure 5.15: Load vs. burst loss ratio, bandwidth utilization, normalized throughput and burst losses in different classes which are calculated by using 1 Mb streams with equal load, 6 wavelengths & periodic streaming with LAUC.**

In figure 5.16, we have presented same results as were in 1 by increasing wavelengths. Here wavelength is increased from 6 to 12. If we compare figure 5.12(a) with figure 5.16(a), then it

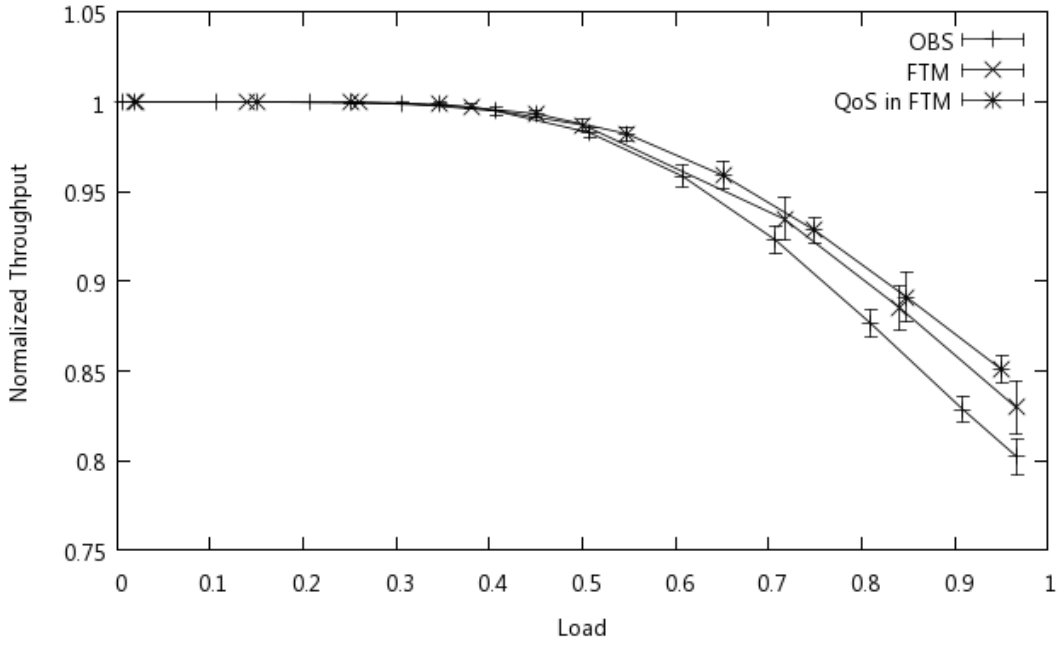
can be seen that burst loss ratio is decreased by increasing wavelengths. Here overall network performance is further improved by adding more data channels to the network.



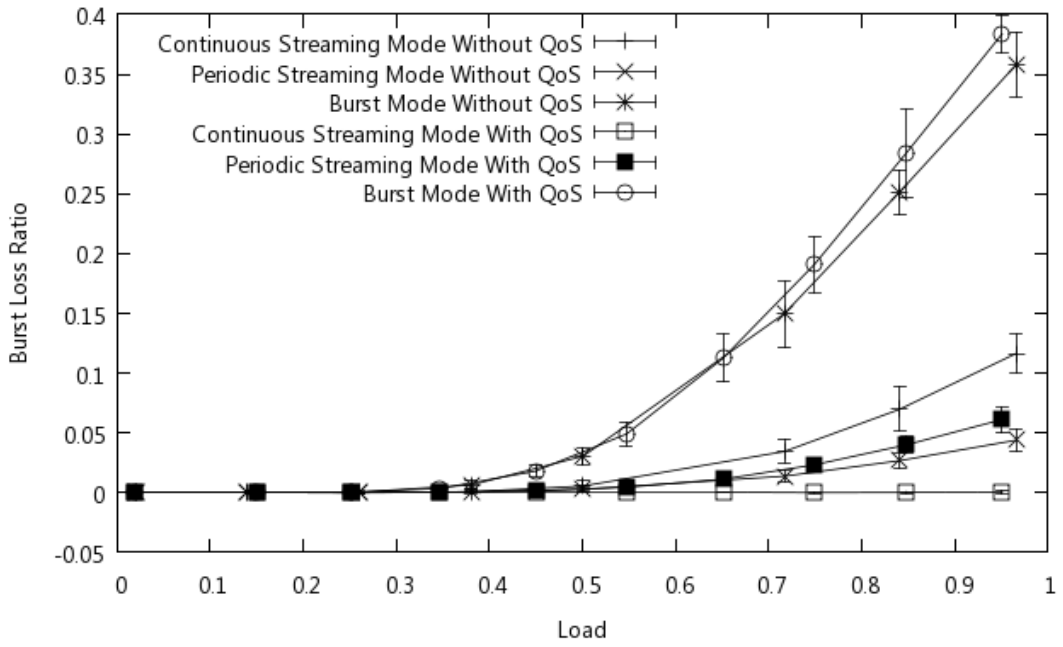

(a) Load vs Burst Loss Ratio




(b) Load vs Bandwidth Utilization




(c) Load vs Normalized Throughput

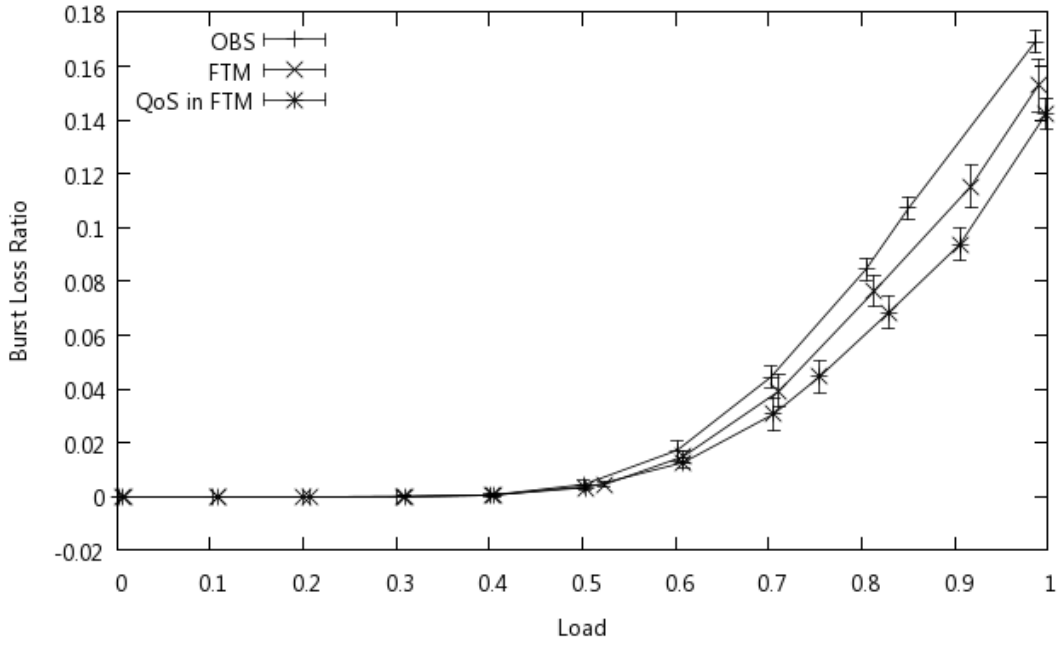



(d) FTM Modes with & without QoS Provisioning

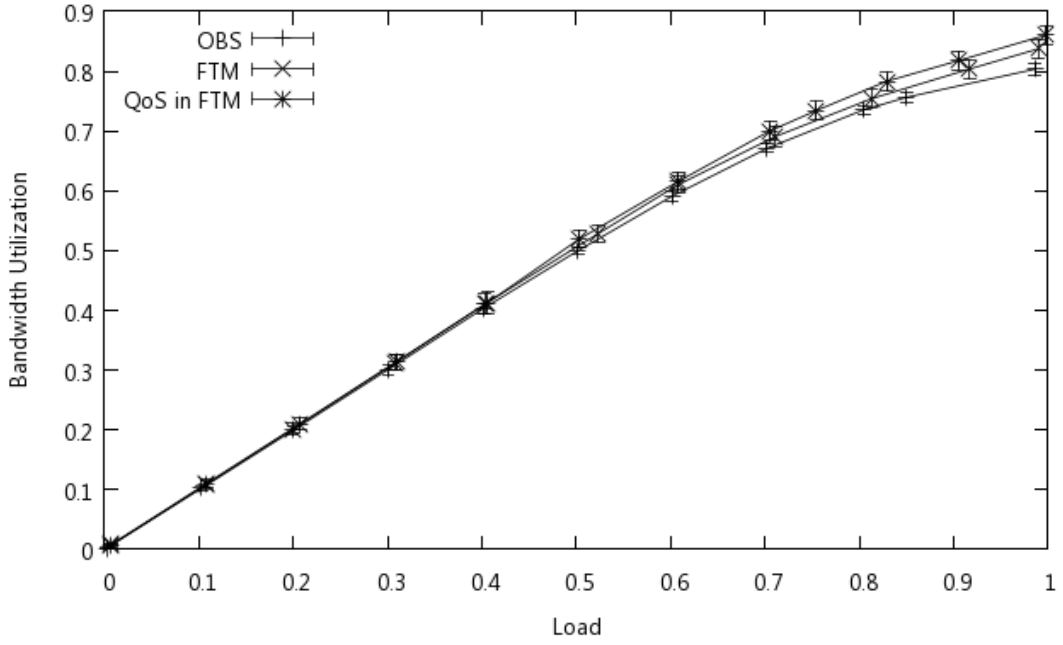
**Figure 5.16: Load vs. burst loss ratio, bandwidth utilization, normalized throughput and burst losses in different classes which are calculated by using 1Mb streams with equal load, 12 wavelengths & periodic streaming with LAUC-VF.**

Again in figure 5.17, we have presented same results as were in case 1 by increasing wavelengths. Here wavelength is increased from 6 to 18. If we compare figure 5.12(a) with figure 5.16(a), then it can be seen that burst loss ratio is decreased by increasing wavelengths.

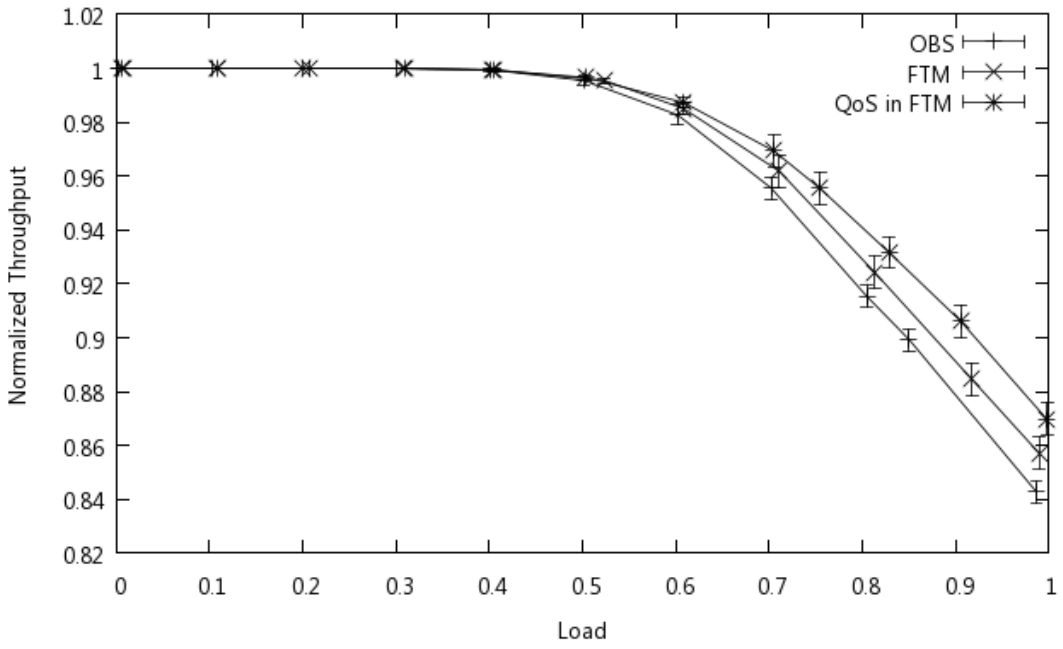
Here overall network performance is further improved by adding more data channels to the network.



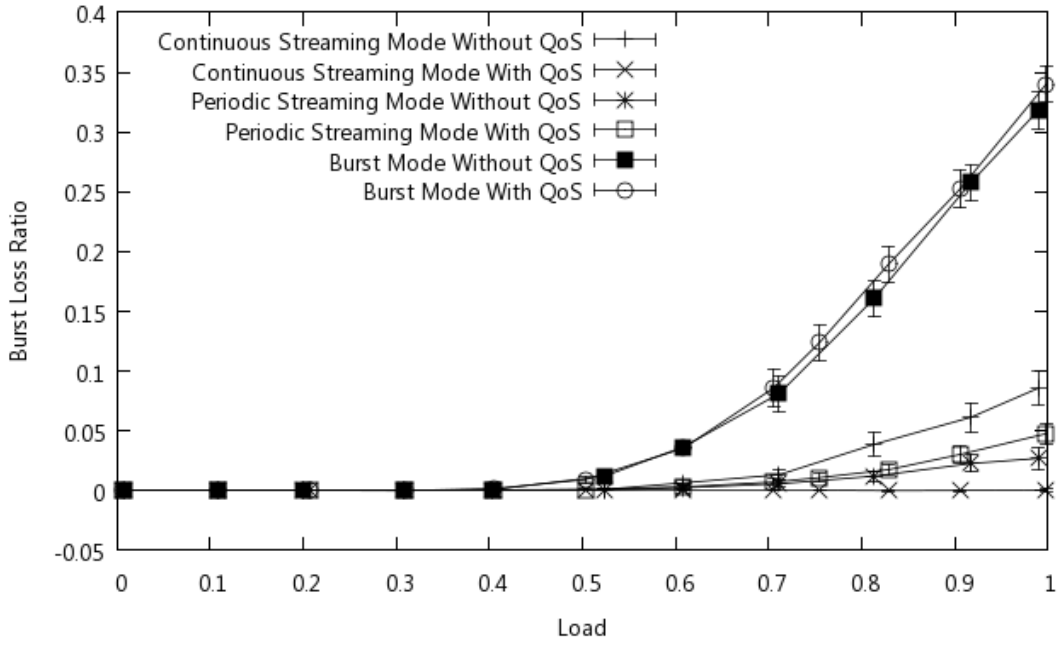

(a) Load vs Burst Loss Ratio




(b) Load vs Bandwidth Utilization




(c) Load vs Normalized Throughput




(d) FTM Modes with & without QoS Provisioning

**Figure 5.17 : Load vs. burst loss ratio, bandwidth utilization, normalized throughput and burst losses in different classes which are calculated by using 1Mb streams with equal load, 18 wavelengths & periodic streaming with LAUC-VF.**

Finally we can see that in all cases, FTM has better performance in terms of burst loss ratio, bandwidth utilization and normalized throughput and performance of FTM is further improved by employing QoS provisioning mechanisms in FTM.

# 6. Summary and Conclusions

## 6.1 Conclusions

Optical networks are ultimate choice for increasing demand of bandwidth hungry applications due to huge bandwidth support. OCS, OPS and OBS are some of the switching techniques available in optical networks. All techniques have some limitations and advantages over one another. OCS has round trip delay and bandwidth under-utilization issue. OPS has limitation of unavailability of appropriate optical RAM as well as output port contention, and OBS has problem of burst losses and throughput maximization.

In order to overcome deficiencies in current switching techniques, an integrated type f approach was proposed named as Flow Transfer Mode. Flow Transfer Mode (FTM) is a universal switching technology based on the theory of OBS and is regarded as the extension / generalization of OBS. FTM integrates all switching mechanisms in one single technique. FTM classifies traffic to different modes. The applications decide which mode should be used for incoming traffic. The big advantages of FTM seem to be lie in case of multimedia streaming traffic due to constant network end to end delay.

Since FTM was the generic idea. There is need for the FTM to be implemented and evaluated its performance. In this thesis, we have implemented FTM using simulation and evaluated its performance. Performance evaluation has been done using comparative analysis of FTM with OBS. We have simulated both OBS and FTM under similar circumstances. Our results show that FTM has low burst loss ratio and more normalized throughput as compared to OBS. There is also improvement in bandwidth utilization in FTM as compared to OBS.

In our results, we find that performance of FTM is improved due to its streaming modes. Also the streaming modes are specially used for multimedia streaming traffic. Multimedia streaming traffic generally requires QoS provisioning. So there is need for the employment of QoS provisioning in FTM. So in this thesis, we have also proposed QoS provisioning for FTM to further improve its performance. Our proposed technique shows improvement as compared to

both FTM and OBS in terms of burst loss ratio, bandwidth utilization and normalized throughput.

## 6.2 Future Work

This research work is based on the performance evaluation of FTM using simulation. Performance evaluation can be done using analytical modeling of FTM.

We have considered burst losses as the main parameter for QoS metrics, for future directions of this work, delay can also be calculated using two way signaling protocols.

We have assumed fixed length of continuous and periodic streams. For future directions, dynamic approach can be devised to dynamically choose a length of streams.

We have used full preemptive dropping technique, for future directions, partial pre-emptive dropping of lower priority bursts can be employed.

# References


[1] Document available at http://www.axiom.fr/upload/fiber-techno.pdf

[2] L. Xu, H.G. Perros, and G Rouskas, “Techniques for optical packet switching and optical burst switching” IEEE Communications Magazine, pp.136-142, Jan. 2001.

[3] C. Qiao and M. Yoo, “Optical Burst Switching (OBS) - A New Paradigm for an Optical Internet,” Journal of High Speed Networks, vol. 8, no.1, pp. 69-84, Jan. 1999.

[4] Yang Chen, Chunming Qiao, Xiang Yu; “Optical burst switching: a new area in optical networking research”, Journal, IEEE Network, vol 18, no.3, May-June 2004, pp. 16-23

[5] Ph.D thesis of Miroslaw Klinkowski “Offset Time-Emulated Architecture for Optical Burst Switching - Modelling and Performance Evaluation”.

[6] Chlamtac, A. Ganz, and G. Karmi. Lightpath communications:An approach to high-bandwidth optical wans. IEEE Transactions on Communications, 40(7):1171{1182, July 1992.

[7] M. Veeraraghavan, R. Karry, T. Moors, M. Karol, and R. Grobler. Architectures and protocols that enable new applications on optical networks. IEEE Communications Magazine, 39(3):118{127, March 2001.

[8] L. Xu, H.G. Perros, and G. Rouskas. Techniques for optical packet switching and optical burst switching. IEEE Communications Magazine, 39(1):136{142, January 2001.

[9] Harmen R. van As “Flow Transfer Mode as Generalization of Optical Burst Switching (OBS)” Networks & Optical Communications NOC-2008.

[10] Harmen R. van As “Time for a Change in Electronic and Photonic Switching”, Transparent Optical Network, ICTON-2008. Vol- 1, pp.140-143.

[11] Harmen R. van As “Design of flow transfer mode switches for generalized optical burst switching” International Workshop on Optical Burst/Packet Switching WOBS2008.

[12] Harmen R. van As “Flow Transfer Mode (FTM) as Universal Switching Method in Electronic and Photonic Networks ” IEEE Conference of Local Computer Networks LCN 2008.

[13] Harmen R. van As “Driving Optical Network Innovation by Extensively Using Transparent Domains” IEEE Conference Transparent Optical Network, ICTON-2010.

[14] Final Draft of Usman Afzal and Talha Imran “QoS in Bimodal Burst Switching” NUST-SEECS Students.

[15] http://en.wikipedia.org/wiki/Optical_ burst_switching

[16] Theoretical Background Chapter on Optical Burst Switching, From thesis of Mr. Faiz Hussain Rizvi, MS(IT) Student at NUST-SEECS.

[17] D. L. Mills et al, “Highball: A High Speed, Reserved-Access, Wide-Area Network," Tech. rep. W-9-3. Elec Eng. Dept., Univ. of Delaware, 1990.

[18] G. C. Hudek and D. J. Muder, "Signalling Analysis for a Multi-Switch All-Optical Network," Proc. IEEE ICC, vol . 2, 1995, pp. 1206-10.

[19 ] Y. Xiong, M. Vandenhoute, md H. Conkoya, "Control Architecture in Optical Burst switched WDM Network IEEE JSAC, Vol 18. Oct 2000 pp 1838-51.

[20] J. Xu et al., "Efficient Channel Scheduling Algorithms in Optical Burst Switched Networks." Proc . INFOCOM, 2003. vol. 3, pp. 2268-78

[21] M. Yoo and C. Qiao, "Just-Enough-Time (JET): A High Speed Protocol for Bursty Traffic in Optical Networks," Proc. IEEE / LEOS Cod Tech. Global Info. Infrastructure, Aug. 1997, pp. 26-27.

[22] Nail Akar, Ezhan Karasan,Kyriakos G. Vlachos, Emmanouel A. Varvarigos, Davide Careglio, Miroslaw Klinkowski, and Josep Solé-Pareta “A survey of quality of service differentiation mechanisms for optical burst switching networks” journal of sciencedirect 2010.

[23] M. Nandi, A. K. Turuk, D. K. Puthal and S. Dutta “Best Fit Void Filling Algorithm in Optical Burst Switching Networks” Second International Conference on Emerging Trends in Engineering and Technology, ICETET-09.

# Appendices

## Appendix A : Poisson Process *

The Poisson Distribution is a discrete distribution which takes on the values X = 0, 1, 2, 3, ... . It is often used as a model for the number of events in a specific time period, e.g. the number of bursts request arriving at core node within given timeframe.

**Probability Mass Function**

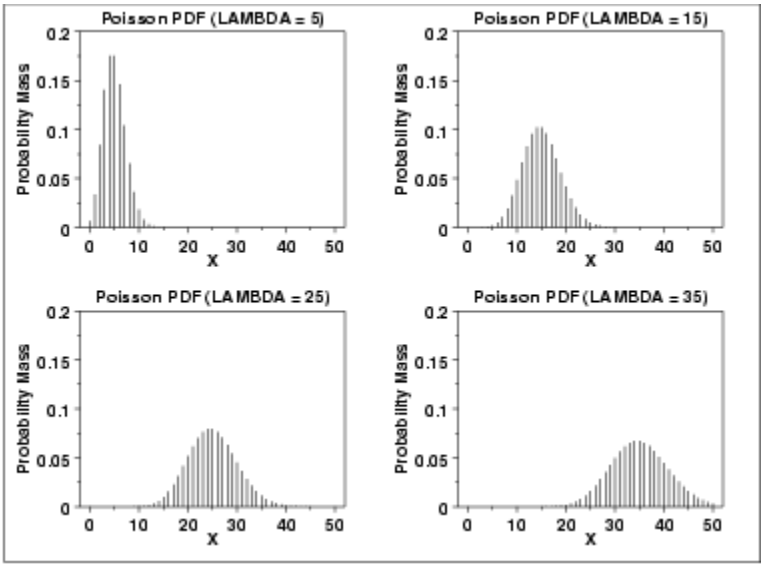
The Poisson distribution is determined by one parameter, lambda. The distribution function for the Poisson distribution is

The formula for the Poisson probability mass function is

$$p(x, \lambda) = \frac{e^{-\lambda}\lambda^x}{x!} \qquad \text{for } x = 0, 1, 2, \cdots$$

$\lambda$ is the shape parameter which indicates the average number of events in the given time interval.

The following is the plot of the Poisson probability density function for four values of $\lambda$.



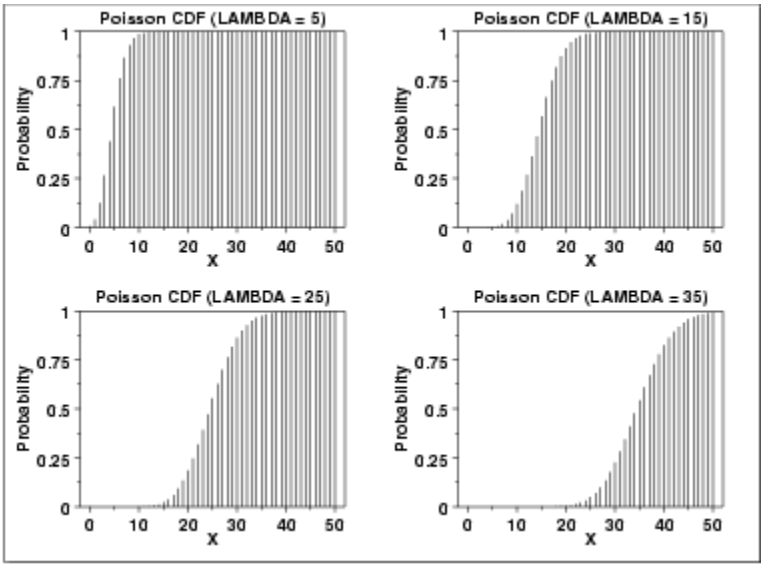

**Cumulative Distribution Function**

The formula for the Poisson cumulative probability function is

$$F(x, \lambda) = \sum_{i=0}^{x} \frac{e^{-\lambda}\lambda^i}{i!}$$

* source: http://www.itl.nist.gov/div898/handbook/eda/section3/eda366j.htm
http://www.math.csusb.edu/faculty/stanton/probstat/poisson.html

The following is the plot of the Poisson cumulative distribution function with the same values of $\lambda$ as the pdf plots above.




**Common Statistics**

| | |
|---|---|
| Mean | $\lambda$ |
| Mode | For non-integer $\lambda$, it is the largest integer less than $\lambda$. For integer $\lambda$, x = $\lambda$ and x = $\lambda$ - 1 are both the mode. |
| Range | 0 to positive infinity |
| Standard Deviation | $\sqrt{\lambda}$ |
| Coefficient of Variation | $\frac{1}{\sqrt{\lambda}}$ |
| Skewness | $\frac{1}{\sqrt{\lambda}}$ |
| Kurtosis | $3 + \frac{1}{\lambda}$ |

**Parameter Estimation**

The maximum likelihood estimator of $\lambda$ is $\tilde{\lambda} = \bar{X}$ where $\bar{X}$ is the sample mean.

## Appendix B: Exponential Distribution*

In probability theory and statistics, the exponential distribution (a.k.a. negative exponential distribution) is a family of continuous probability distributions. It describes the time between events in a Poisson process, i.e. a process in which events occur continuously and independently at a constant average rate. In our case, it is used to calculate inter-arrival time of bursts and also used to calculate burst length.

**Probability Density Function**

The general formula for the probability density function of the exponential distribution is

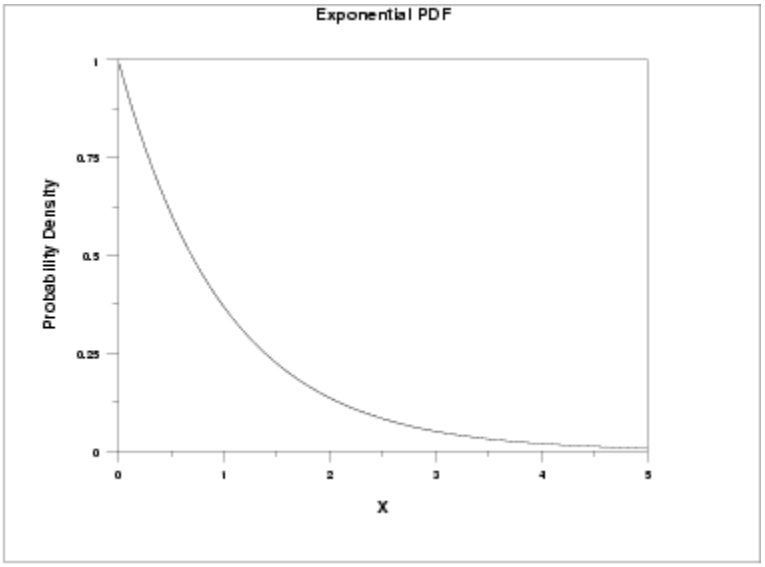
$$f(x) = \frac{1}{\beta} e^{-(x-\mu)/\beta} \qquad x \geq \mu; \beta > 0$$

where $\mu$ is the location parameter and $\beta$ is the scale parameter(the scale parameter is often referred to as $\lambda$ which equals $1/\beta$). The case where $\mu = 0$ and $\beta = 1$ is called the standard exponential distribution. The equation for the standard exponential distribution is

$$f(x) = e^{-x} \qquad \text{for } x \geq 0$$

The general form of probability functions can be expressed in terms of the standard distribution. Subsequent formulas in this section are given for the 1-parameter (i.e., with scale parameter) form of the function.

The following is the plot of the exponential probability density function.



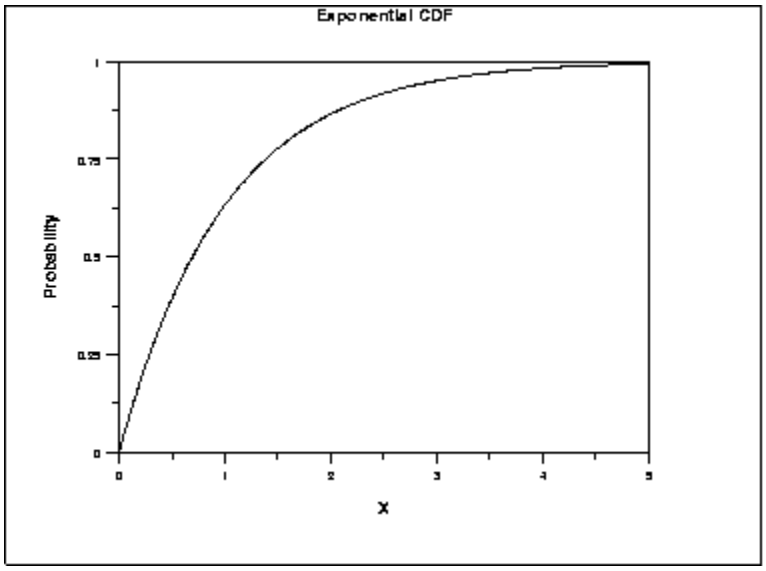

*source: http://itl.nist.gov/div898/handbook/eda/section3/eda3667.htm
http://en.wikipedia.org/wiki/Exponential_distribution

## Cumulative Distribution Function

The formula for the cumulative distribution function of the exponential distribution is

$$F(x) = 1 - e^{-x/\beta} \qquad x \geq 0; \beta > 0$$

The following is the plot of the exponential cumulative distribution function.




## Common Statistics

| | |
|---|---|
| Mean | $\beta$ |
| Median | $\beta \ln 2$ |
| Mode | Zero |
| Range | Zero to plus infinity |
| Standard Deviation | $\beta$ |
| Coefficient of Variation | 1 |
| Skewness | 2 |
| Kurtosis | 9 |

# Appendix C : Simulation Code

```
// This class is used for simulation of OBS Network.
import java.util.*;
import java.text.DecimalFormat;
import java.io.*;
/*
@Author Muhammad Imran
*/
public class OBSSimulation{
	public static double meanload=0, meanbandwidth=0, meanlossrate=0, meaninterarrivaltime=0, meanloadinper=0, inter=0,
	mean_length=0;
	public static double indload=0, indband=0, indloss=0, indthroughput=0, bitspassed=0, bitsdropped=0,
	indnormalizedthroughput=0;
	int controlchannels=1;
	double datarate = 10000000000.00;
	int nodes = 4;
	int wv=18;		// Total number of wavelength per node
	public static Random r; // Random number generator
	public static Random r1; // Random number generator
	double mean_burst_size = 50*1000*8;
	double[] utilization = new double [wv]; // Bandwidth utilization
	double[] wv_utilization = new double [wv]; // Bandwidth used per node
	ArrayList<ArrayList<ArrayList>> channel_list = new ArrayList<ArrayList<ArrayList>>();
public void runSimulation(double mean_arrival_rate){
	try{
		r = new Random();
		r1 = new Random();
		double simulation_time = 100000;
		meanload=0;meanbandwidth=0; meanlossrate=0; meaninterarrivaltime=0; meanloadinper=0; inter=0; mean_length=0;
		indload=0; indband=0; indloss=0; indthroughput=0; bitspassed=0; bitsdropped=0;	indnormalizedthroughput=0;

		for (int x=0; x<wv; x++)
			{
			wv_utilization[x] = 0.0;
			utilization[x] = 0.0;
			}
		channel_list.clear();
		for (int i=0;i<wv;i++)
		{
			channel_list.add(i,new ArrayList<ArrayList>());
		}
		LinkedList eventlist=new LinkedList(); //create event list
		LinkedList schedule_burst = new LinkedList();
		LinkedList unschedule_burst = new LinkedList();
		int counter=0;
		node n1=null;
		int MCL = 0;
		Comp2 comp = new Comp2();
		Collections.sort(eventlist,comp);
		int propogaration_delay = 1000;
		int configuration_delay = 3;
		int burst_id=0;
		int cp_id=0;
		int dest = 3;
		double total_utilization=0;
		while (counter<simulation_time){

			int source = (int)Math.round(r.nextDouble()*(nodes-2));
			double inter_arrival_time = calculate_arrival_time(mean_arrival_rate);
			MCL += inter_arrival_time;
			meaninterarrivaltime += inter_arrival_time;
			double burst_size = calculate_burst_size(mean_burst_size);
			double burst_duration = Math.round((burst_size)/10000);
```

```
                CP cp = new CP(0,      source, dest, propogaration_delay,configuration_delay, MCL,
                MCL+propogaration_delay,MCL+propogaration_delay+inter_arrival_time, cp_id+"p",burst_duration);
                node n2 = new node(2,cp.getArrival_time(), 1, cp);
                n2.setBhc(cp);
                eventlist.add(n2);
                cp_id++;
                burst_id++;
                counter+=inter_arrival_time;
        }
        Collections.sort(eventlist,comp);
        int size = eventlist.size();
        n1= (node) eventlist.get(0);
        int count = 0;
        double master_clock=0;
        double initialclock = n1.getBhc().getOffset_time();
        while (count<size)
                {
                Collections.sort(eventlist,comp);
                n1= (node) eventlist.get(count);
                master_clock= n1.getClock();
                double periority =  n1.getBhc().getPeriority();
                double start = n1.getBhc().getOffset_time();
                double end = start + n1.getBhc().getBurst_duration();
                boolean found = scheduleChannel(start, end,periority);
                if (found){
                        schedule_burst.add(n1.getBhc().getBhc_id());
                        bitspassed += (n1.getBhc().getBurst_duration()*10000);
                        }
                else{
                        unschedule_burst.add(n1.getBhc().getBhc_id());
                        bitsdropped += (n1.getBhc().getBurst_duration()*10000);
                        }
                        eventlist.remove(count);
                        size = eventlist.size();
                }
                double totalBurst = schedule_burst.size()+ unschedule_burst.size();
                double unschburst = unschedule_burst.size();
                double schburst = schedule_burst.size();


                for (int i=0;i<wv;i++)
                {
                total_utilization += wv_utilization[i];
                }
                total_utilization  = (total_utilization / (wv));
                double bandwidth_util = total_utilization / master_clock;
                double lossratio =  (unschburst / totalBurst)  ;
                indloss = lossratio;
                meanload += schedule_burst.size()+unschedule_burst.size();
                meanlossrate += lossratio;
                meanbandwidth +=bandwidth_util;
                indnormalizedthroughput = schburst/totalBurst;
                inter += meaninterarrivaltime/meanload;
                double bitsthroughput = bitspassed / (bitspassed+bitsdropped);
                bitspassed = bitspassed / (1000 * 1000 *1000);
                simulation_time = simulation_time / (1000 * 1000);
                indthroughput = ((bitspassed / simulation_time) )/wv;//  G bits per second
                indband = (indthroughput / 10) ; // percentage
                mean_length += mean_burst_size/10000;
                double load = (1/(meaninterarrivaltime/meanload)) / (1/(mean_burst_size/10000));
                meanloadinper +=(load/wv);
                indload = (load/wv);
                }catch(Exception e){

System.out.println("Exception"+e.getMessage()+e.getLocalizedMessage());
                        }
```

```
                                        }
public boolean scheduleChannel(double starttime, double endtime,double periority){
            int channel=0;
            boolean schedule = false;
            boolean scheduleusingvoid = false;
            boolean scheduleusinglauc = false;

            try{
            double[] startingvoid = new double[wv];
            double[] endingvoid = new double[wv];
            for (int i=0;i<wv;i++){
            if (channel_list.get(i).size() > 0){
            double startingdiff = 0;
            double endingdiff = 0;
            for (int j=0;j<channel_list.get(i).size();j++){
            if(starttime > Double.parseDouble (channel_list.get(i).get(j).get(2).toString())){
            if (Double.parseDouble (channel_list.get(i).get(j).get(2).toString()) > startingdiff)
            {
            startingdiff =  Double.parseDouble(channel_list.get(i).get(j).get(2).toString());
            startingvoid[i] = starttime- Double.parseDouble(channel_list.get(i).get(j).get(2).toString());
            }
            }
            }
            double temp = 999999999999.0;
            for (int j=0;j<channel_list.get(i).size();j++){
            if(Double.parseDouble(channel_list.get(i).get(j).get(1).toString())>startingdiff){
            if(Double.parseDouble(channel_list.get(i).get(j).get(1).toString())<temp){
            temp = Double.parseDouble(channel_list.get(i).get(j).get(1).toString());
                        }
            }
            if(temp==999999999999.0)
            {

            }else{
                        endingvoid[i]=temp-endtime;
                                                }
                        }
            }

            double minvoid = startingvoid[0];
            boolean foundvoid=false;
            for (int i=0;i<wv;i++){
            if(startingvoid[i]<=minvoid && endingvoid[i]>0.0  && startingvoid[i]>0.0){
                        minvoid = startingvoid[i];
                        foundvoid=true;
                        channel = i;
                        schedule=true;
                        }
            }
            if(foundvoid){
            channel_list.get(channel).add(channel_list.get(channel).size(),new ArrayList());
            channel_list.get(channel).get(channel_list.get(channel).size()-1) .add(periority);
            channel_list.get(channel).get(channel_list.get(channel).size()-1) .add(starttime);
            channel_list.get(channel).get(channel_list.get(channel).size()-1) .add(endtime);
            wv_utilization[channel]+=(endtime-starttime);
            }else{
                        boolean foundusinglauc=false;
                        double[] laucend = new double[wv];
                        for (int i=0;i<wv;i++){
                                    if (channel_list.get(i).size() > 0){
                                                double maxlauc =0;
                                                for (int j=0;j<channel_list.get(i).size();j++){
            if(Double.parseDouble(channel_list.get(i).get(j).get(2).toString())>maxlauc )

                        maxlauc=Double.parseDouble(channel_list.get(i).get(j).get(2).toString());
                                                }
```

```
                                        laucend[i]=maxlauc;
                                }
                        }
                        double maxlauc =0;
                        for (int i=0;i<wv;i++){
                                if(laucend[i]>=maxlauc  && laucend[i]<starttime ){
                                        maxlauc = laucend[i];
                                        foundusinglauc=true;
                                        channel = i;
                                }
                        }
                        if(foundusinglauc){
                                schedule=true;
                                channel_list.get(channel).add(channel_list.get(channel).size(),new ArrayList());
                                channel_list.get(channel).get(channel_list.get(channel).size()-1).add(periority);
                                channel_list.get(channel).get(channel_list.get(channel).size()-1).add(starttime);
                                channel_list.get(channel).get(channel_list.get(channel).size()-1).add(endtime);
                                wv_utilization[channel]+=(endtime-starttime);
                                }
                        else{
                        for (int i=0;i<wv;i++){
                        if (channel_list.get(i).size()==0){
                        channel = i;
                        channel_list.get(channel).add(0,new ArrayList());
                        channel_list.get(channel).get(0).add(periority);
                        channel_list.get(channel).get(0).add(starttime);
                        channel_list.get(channel).get(0).add(endtime);
                        schedule = true;
                        wv_utilization[channel]+=(endtime-starttime);
                        System.out.println("Channel : "+channel+"  "+ channel_list.get(channel));
                        break;
                        }

                }
        }
}
}
catch(Exception e ){
        System.out.println(e.getMessage());
}
return schedule;
}
public  double calculate_arrival_time(double mean){
        double random=r.nextDouble();
        double exp_time=(-1/mean)*Math.log(random);
        return Math.round(exp_time);
}

public  double calculate_burst_size(double mean){
        double random=r1.nextDouble();
        double exp_time=(-mean)*Math.log(random);
        return Math.round(exp_time);
        }
public static void main(String agrs[])
{
        OBSSimulation f = new OBSSimulation();
        double mean_arrival_rate = 0.38;
        for (double m = mean_arrival_rate; mean_arrival_rate<=0.451;
        mean_arrival_rate+=0.045){

        int samplesize = 20;
        double[] loss = new double[samplesize];
            double[] band = new double[samplesize];
            double[] load = new double[samplesize];
            double[] throughput = new double[samplesize];
            double[] northroughput = new double[samplesize];
```

```
            double losssum=0, bandsum=0, loadsum=0, throughputsum=0,  northroughputsum=0;
            try {
                        FileWriter fstream = new FileWriter("obs18.txt",true);
                        BufferedWriter out = new BufferedWriter(fstream);
 for(int i=0; i<samplesize;i++){
            f.runSimulation(mean_arrival_rate);
            load[i]=indload;
            loadsum+=load[i];
            loss[i]=indloss;
            losssum+=loss[i];
            band[i]=indband;
            bandsum+=band[i];
            throughput[i]=indthroughput;
            throughputsum+=throughput[i];
            northroughput[i]=indnormalizedthroughput;
            northroughputsum+=northroughput[i];
}
 double meanloss = losssum / samplesize;
 double meanband= bandsum / samplesize;
 double meanload= loadsum / samplesize;
 double meanthrouput= throughputsum / samplesize;
 double meannorthrouput= northroughputsum / samplesize;
 System.out.println("Mean Load "+meanload);
 System.out.println("Mean Band" +meanband);
 System.out.println("Mean Throughput" +meanthrouput);
 System.out.println("Mean Normalized Throughput" +meannorthrouput);
            double xxmeansum = 0.0,xxmeanband=0.0,xxmeanthroughput=0.0,  xxmeannorthroughput=0;
for (int j = 0; j < samplesize; j++) {
xxmeansum += (loss[j] - meanloss) * (loss[j] - meanloss);
xxmeanband += (band[j] - meanband) * (band[j] - meanband);
xxmeanthroughput += (throughput[j] - meanthrouput) * (throughput[j] - meanthrouput);
xxmeannorthroughput += (northroughput[j] - meannorthrouput) * (northroughput[j] - meannorthrouput);
   }
   double varianceloss = xxmeansum / (samplesize - 1);
   double varianceband = xxmeanband / (samplesize - 1);
   double variancethroughput = xxmeanthroughput / (samplesize - 1);
   double variancenorthroughput = xxmeannorthroughput / (samplesize - 1);
   double stddevloss = Math.sqrt(varianceloss);
   double stddevband = Math.sqrt(varianceband);
   double stddevthroughput = Math.sqrt(variancethroughput);
   double stddevnorthroughput = Math.sqrt(variancenorthroughput);
   double loloss = meanloss - (1.96 * stddevloss);
   double hiloss = meanloss + (1.96 * stddevloss);
   double loband = meanband - (1.96 * stddevband);
   double hiband = meanband + (1.96 * stddevband);
   double lothroughput = meanthrouput - (1.96 * stddevthroughput);
   double hithroughput = meanthrouput + (1.96 * stddevthroughput);

   double lonorthroughput = meannorthrouput - (1.96 * stddevnorthroughput);
   double hinorthroughput = meannorthrouput + (1.96 * stddevnorthroughput);

   System.out.println("average    loss    = " + meanloss);
            System.out.println("sample variance loss = " + varianceloss);
            System.out.println("sample stddev loss   = " + stddevloss);
            System.out.println("95% approximate confidence interval");
            System.out.println("[ " + (loloss-meanloss) + ", " + (hiloss-meanloss) + " ]");

            System.out.println("average    band    = " + meanband);
            System.out.println("sample variance band = " + varianceband);
            System.out.println("sample stddev band   = " + stddevband);
            System.out.println("95% approximate confidence interval");
            System.out.println("[ " + (loband-meanband) + ", " + (hiband-meanband) + " ]");

   out.newLine();
   out.write(Double.toString(meanload));
   out.write("            ");
```

```
                out.write(Double.toString(meanloss));
                out.write("               ");
                out.write(Double.toString((hiloss-meanloss)));
                out.write("               ");
                out.write(Double.toString(meanband));
                out.write("               ");

                out.write(Double.toString((hiband-meanband)));
                out.write("               ");
                out.write(Double.toString(meanthrouput));
                out.write("               ");
                out.write(Double.toString((hithroughput-meanthrouput)));

                out.write("               ");
                out.write(Double.toString(meannorthrouput));
                out.write("               ");
    out.write(Double.toString((hinorthroughput-meannorthrouput)));
    out.close();

}catch(Exception e){
                System.out.println("Geo "+ e.getLocalizedMessage());
}
finally {
//  write.close();
                                }


                                }

                }
}
// This class is used for sorting of event list
class Comp2 implements Comparator {
                 public int compare (Object n1, Object n2) {

                   if(((node)n1).getClock() > ((node)n2).getClock())
                                return 1;
                   else if(((node)n1).getClock() == ((node)n2).getClock()) return 0;
                   else return -1;


                                }
}
// this class gives the variables and methods for each event in the event list
package FTMSimulation;
/*
@Author Muhammad Imran
*/
public class node{
                int eventNo; // event number is 1 for arrival , 2 for departure
                double clock;             // time at which the evnt will occur
                int type; // 1 for control signal and 2 for burst
                int[] path;  // stores the wavelengths on a path from source to destination
                double arrival_time;
                long size;
                Burst b ;
                int current_node;                       //
                int current_node_type; // 0 for edge and 1 for core node
                CP bhc ;
                boolean schedule = false;
                public node ()
                {
                }
                public node(int eventNo, double clock, int type, CP bhc) {
                                super();
                                this.eventNo = eventNo;
```

```
		this.clock = clock;
		this.type = type;
		this.bhc = bhc;
		current_node= bhc.getSource();
		current_node_type=0;
	}
	public node(int eventNo, double clock, int type, Burst b) {
		super();
		this.eventNo = eventNo;
		this.clock = clock;
		this.type = type;
		this.b = b;
		current_node= b.getSource();
		current_node_type=0;
	}

	public node(int m,int c, int type){
		this.eventNo=m;
		this.clock=c;
		this.type=type;
	}
	public int getEventNo() {
		return eventNo;
	}
	public void setEventNo(int eventNo) {
		this.eventNo = eventNo;
	}
	public double getClock() {
		return clock;
	}
	public void setClock(double clock) {
		this.clock = clock;
	}
	public int getType() {
		return type;
	}
	public void setType(int type) {
		this.type = type;
	}
	public int[] getPath() {
		return path;
	}
	public void setPath(int[] path) {
		this.path = path;
	}
	public double getArrival_time() {
		return arrival_time;
	}
	public void setArrival_time(double arrival_time) {
		this.arrival_time = arrival_time;
	}
	public long getSize() {
		return size;
	}
	public void setSize(long size) {
		this.size = size;
	}
	public Burst getB() {
		return b;
	}
	public void setB(Burst b) {
		this.b = b;
	}
	public int getCurrent_node() {
		return current_node;
	}
```

```
	public void setCurrent_node(int current_node) {
		this.current_node = current_node;
	}
	public int getCurrent_node_type() {
		return current_node_type;
	}

	public void setCurrent_node_type(int current_node_type) {
		this.current_node_type = current_node_type;
	}
	public CP getBhc() {
		return bhc;
	}
	public void setBhc(CP bhc) {
		this.bhc = bhc;
	}
	public boolean isSchedule() {
		return schedule;
	}
	public void setSchedule(boolean schedule) {
		this.schedule = schedule;
	}
}
// this class is used for maintaining states of control packets
package FTMSimulation;
/*
@Author Muhammad Imran
*/
public class CP {
int source=0, destination=0,propogation_delay=0,configuration_time=0, periority=0;
double departure_time=0, arrival_time=0,burst_duration=0,interarrivaltime=0;
double offset_time=0;
String bhc_id=null;
public CP(int periority, int source, int destination, int propogation_delay,
int configuration_time, double departure_time, double arrival_time,
double offset_time, String bhc_id, double burst_duration) {
super();
	this.periority=periority;
	this.source = source;
	this.destination = destination;
	this.propogation_delay = propogation_delay;
	this.configuration_time = configuration_time;
	this.departure_time = departure_time;
	this.arrival_time = arrival_time;
	this.offset_time = offset_time;
	this.bhc_id = bhc_id;
	this.burst_duration= burst_duration;
}
public CP(int periority, int source, int destination, int propogation_delay,
		int configuration_time, double departure_time, double arrival_time, double offset_time, String bhc_id, double
burst_duration, double interarrivaltime) {
		super();
		this.periority=periority;
		this.source = source;
		this.destination = destination;
		this.propogation_delay = propogation_delay;
		this.configuration_time = configuration_time;
		this.departure_time = departure_time;
		this.arrival_time = arrival_time;
		this.offset_time = offset_time;
		this.bhc_id = bhc_id;
		this.burst_duration= burst_duration;
		this.interarrivaltime= interarrivaltime;
	}
	public double getInterarrivaltime() {
		return interarrivaltime;
```

```
	}
	public void setInterarrivaltime(double interarrivaltime) {
		this.interarrivaltime = interarrivaltime;
	}
	public double getBurst_duration() {
		return burst_duration;
	}
	public void setBurst_duration(double burst_duration) {
		this.burst_duration = burst_duration;
	}
	public int getSource() {
		return source;
	}
	public void setSource(int source) {
		this.source = source;
	}
	public int getDestination() {
		return destination;
	}
	public void setDestination(int destination) {
		this.destination = destination;
	}
	public int getPropogation_delay() {
		return propogation_delay;
	}
	public void setPropogation_delay(int propogation_delay) {
		this.propogation_delay = propogation_delay;
	}
	public int getConfiguration_time() {
		return configuration_time;
	}
	public void setConfiguration_time(int configuration_time) {
		this.configuration_time = configuration_time;
	}
	public double getDeparture_time() {
		return departure_time;
	}
	public void setDeparture_time(double departure_time) {
		this.departure_time = departure_time;
	}
	public double getArrival_time() {
		return arrival_time;
	}
	public void setArrival_time(double arrival_time) {
		this.arrival_time = arrival_time;
	}
	public double getOffset_time() {
		return offset_time;
	}
	public void setOffset_time(double offset_time) {
		this.offset_time = offset_time;
	}
	public String getBhc_id() {
		return bhc_id;
	}
	public void setBhc_id(String bhc_id) {
		this.bhc_id = bhc_id;
	}
	public int getPeriority() {
		return periority;
	}
	public void setPeriority(int periority) {
		this.periority = periority;
	}
}
// this class is used for bursts
```

```
package FTMSimulation;
/*
@Author Muhammad Imran
*/
public class Burst {
String burst_id;
long size;                                  //burst size in bytes
int packets;                                //how much packets to include rom queue
double departure_time;                                          //when will the packet be ready for transmission
        int propagation_time;
        int source;
        int destination;
        double arrival_time;
        double burst_duration;
        boolean schedule = false;
        public Burst(String burst_id, double departure_time,
        int propagation_time, int source, int destination,
                double arrival_time, double burst_duration) {
                super();
                        this.burst_id = burst_id;
                        this.departure_time = departure_time;
                        this.propagation_time = propagation_time;
                        this.source = source;
                        this.destination = destination;
                        this.arrival_time = arrival_time;
                        this.burst_duration = burst_duration;
                }
                public String getBurst_id() {
                        return burst_id;
                }
                public void setBurst_id(String burst_id) {
                        this.burst_id = burst_id;
                }
                public long getSize() {
                        return size;
                }
                public void setSize(long size) {
                        this.size = size;
                }
                public int getPackets() {
                        return packets;
                }
                public void setPackets(int packets) {
                        this.packets = packets;
                }
                public double getDeparture_time() {
                        return departure_time;
                }
                public void setDeparture_time(double departure_time) {
                        this.departure_time = departure_time;
                }
                public int getPropagation_time() {
                        return propagation_time;
                }
                public void setPropagation_time(int propagation_time) {
                        this.propagation_time = propagation_time;
                }
                public int getSource() {
                        return source;
                }
                public void setSource(int source) {
                        this.source = source;
                }
                public int getDestination() {
                        return destination;
                }
```

```
                        public void setDestination(int destination) {
                                this.destination = destination;
                        }
                        public double getArrival_time() {
                                return arrival_time;
                        }
                        public void setArrival_time(double arrival_time) {
                                this.arrival_time = arrival_time;
                        }
                        public double getBurst_duration() {
                                return burst_duration;
                        }
                        public void setBurst_duration(double burst_duration) {
                                this.burst_duration = burst_duration;
                        }
                        public boolean isSchedule() {
                                return schedule;
                        }
                        public void setSchedule(boolean schedule) {
                                this.schedule = schedule;
                        }

}
// this class is used for simulating FTM Network With QoS Provisioning
package FTMSimulation;
import java.util.*;
import java.text.DecimalFormat;
import java.io.*;
import FTMSimulation.*;
/*
@Author Muhammad Imran

*/
public class FTMWithQos{
public static double
totalbursts=0,meanbandwidth=0,meanlossrate=0,totalinterarrivaltime=0,totalburstssizes=0,meanloadinper=0,inter=0,mean_length=0;
public static double indload=0,indband=0,indloss=0,indthroughput=0,bitspassed=0,bitsdropped=0, indnormalizedthroughput=0, indcont
=0,indper=0,indnorm=0;
public static double indobsload=0,indcontload=0,indperload=0,totalintarrivalcont=0,totalintarrivalper=0, totalperburstsizes=0;
int controlchannels=1;
double datarate = 10000000000.00;
int nodes = 4;
int wv=6;               // Total number of wavelength per node
public static Random r;// Random number generator
public static Random r1; // Random number generator
public static Random r2; // Random number generator
double mean_burst_size = 50*1000*8;
double mean_burst_size_continuous = 500*1000*8;
double[] utilization = new double [wv]; // Bandwidth utilization
double[] wv_utilization = new double [wv]; // Bandwidth used per node
ArrayList<ArrayList<ArrayList>> channel_list = new ArrayList<ArrayList<ArrayList>>();
int burst_schedule = 0;
int burst_dropped =0;
double continuous_bursts_dropped=0,continuous_bursts_successful=0;;
double periodic_bursts_dropped=0,periodic_bursts_successful=0;
double normal_bursts_dropped=0,normal_bursts_successful=0;

public void runSimulation(double mean_arrival_rate){
try{
                        r=new Random(); // Random number generator
                        r1=new Random(); // Random number generator
                        r2=new Random(); // Random number generator
                        double simulation_time = 500000;
totalbursts=0; meanbandwidth=0; meanlossrate=0; totalinterarrivaltime=0 ;meanloadinper=0 ; inter=0; mean_length=0;
burst_schedule = 0;totalburstssizes=0;
burst_dropped =0;
```

```
indthroughput=0;bitspassed=0;bitsdropped=0;
continuous_bursts_dropped=0;continuous_bursts_successful=0;;
periodic_bursts_dropped=0;periodic_bursts_successful=0;
normal_bursts_dropped=0;normal_bursts_successful=0;
indcont =0;indper=0;indnorm=0;
indobsload=0;indcontload=0;indperload=0;totalintarrivalcont=0;
indload=0;indband=0;indloss=0;indthroughput=0;bitspassed=0;bitsdropped=0; indnormalizedthroughput=0; indcont =0;indper=0;indnorm=0;
indobsload=0;indcontload=0;indperload=0;totalintarrivalcont=0;totalintarrivalper=0; totalperburstsizes=0;
indnormalizedthroughput=0;
totalperburstsizes=0;
indload=0;indband=0;indloss=0;
totalintarrivalper=0;
for (int x=0; x<wv; x++)
{
wv_utilization[x] = 0.0;
utilization[x] = 0.0;
}
                                        channel_list.clear();
                                        for (int i=0;i<wv;i++)

                                        {
                                                channel_list.add(i,new ArrayList<ArrayList>());

                                        }
                                        LinkedList eventlist=new LinkedList(); //create event list
                                        int counter=0;
                                        node n1=null;
                                        int MCL = 0;
                                        Comp1 comp = new Comp1();
                                        Collections.sort(eventlist,comp);
                                        int propogaration_delay = 1000;
                                        int configuration_delay = 3;
                                        int cont_burst=10;
                                        int periodic_burst = 10;
                                        int cp_id=1;
                                        int dest = 3;
                                        double total_utilization=0;
                                        int mode =0;
// Obs mode generation
while (counter<simulation_time){
double inter_arrival_time = calculate_arrival_time(mean_arrival_rate);
MCL += inter_arrival_time;
totalinterarrivaltime += inter_arrival_time;
double burst_size = calculate_burst_size(mean_burst_size);
double burst_duration = Math.round((burst_size)/10000);
                                        CP cp;
//              totalburstssizes += burst_duration;
cp = new CP(0,          0, dest, propogaration_delay,configuration_delay, MCL,
MCL+propogaration_delay,MCL+propogaration_delay+inter_arrival_time, cp_id+"p",burst_duration);
                counter+=inter_arrival_time;
                node n2 = new node(2,cp.getArrival_time(), 1, cp);
                n2.setBhc(cp);
                eventlist.add(n2);
                cp_id++;
}

// Continuous Stream Generation
MCL = 0;
counter =0;
while (counter<simulation_time){

double inter_arrival_time = calculate_arrival_time(mean_arrival_rate);

inter_arrival_time = inter_arrival_time * cont_burst;
MCL += inter_arrival_time;
totalintarrivalcont += inter_arrival_time;
```

```
double burst_size = calculate_burst_size(mean_burst_size_continuous);
double burst_duration = Math.round((burst_size)/10000);
double extraoffset = (mean_burst_size/10000) * 4;
CP cp;
cp = new CP(2,2, dest, propogaration_delay,configuration_delay, MCL,
MCL+propogaration_delay,MCL+propogaration_delay+inter_arrival_time+extraoffset, cp_id+"p",burst_duration);
counter+= inter_arrival_time;
node n2 = new node(2,cp.getArrival_time(), 1, cp);
n2.setBhc(cp);
eventlist.add(n2);
cp_id++;
}

// Periodic Stream Generation
MCL = 0;
counter =0;
while (counter<simulation_time){
double inter_arrival_time = calculate_arrival_time(mean_arrival_rate);
inter_arrival_time = inter_arrival_time;
MCL += inter_arrival_time;
totalintarrivalper += inter_arrival_time;
double burst_size = calculate_burst_size(mean_burst_size);
double burst_duration = Math.round((burst_size)/10000);
totalperburstsizes += burst_duration*periodic_burst;
CP cp;
cp = new CP(1,          1, dest, propogaration_delay,configuration_delay, MCL,
MCL+propogaration_delay,MCL+propogaration_delay+(inter_arrival_time), cp_id+"p",burst_duration,inter_arrival_time);
totalintarrivalper += inter_arrival_time*(periodic_burst-1);
counter+= inter_arrival_time*(periodic_burst-1);
MCL+= inter_arrival_time*(periodic_burst-1);
node n2 = new node(2,cp.getArrival_time(), 1, cp);
n2.setBhc(cp);
eventlist.add(n2);
cp_id++;
}
Collections.sort(eventlist,comp);
int size = eventlist.size();
n1= (node) eventlist.get(0);
int count = 0;
double master_clock=0, last_clock=0;
double initialclock = n1.getBhc().getOffset_time();
while (count<size)
{
Collections.sort(eventlist,comp);
n1= (node) eventlist.get(count);
master_clock= n1.getClock();
double start = 0;
double end = 0 ;
double periority =  n1.getBhc().getPeriority();
boolean found =false;
if (periority==1) {
start = n1.getBhc().getOffset_time();
end = start + n1.getBhc().getBurst_duration();
for(int i =0; i<periodic_burst ; i++){
found = scheduleChannel(start, end,periority);
if(found){
        burst_schedule++;
        periodic_bursts_successful++;
        bitspassed += (n1.getBhc().getBurst_duration()*10000);
        if(end > last_clock)
                last_clock = end;
}
else{
        burst_dropped++;
        periodic_bursts_dropped++;
        bitsdropped += (n1.getBhc().getBurst_duration()*10000);
```

```
}
start = end + n1.getBhc().getInterarrivaltime();
end = start + n1.getBhc().getBurst_duration();
}
}
else
if (periority==2) {
		start = n1.getBhc().getOffset_time();
		end = start + n1.getBhc().getBurst_duration();
		found = scheduleChannel(start, end,periority);
if(found){
		burst_schedule +=cont_burst;
		continuous_bursts_successful +=cont_burst;
		bitspassed += (n1.getBhc().getBurst_duration()*10000);
if (end>last_clock) {
		last_clock = end;
}
}
else{
		burst_dropped+=cont_burst;
		continuous_bursts_dropped+=cont_burst;
		bitsdropped += (n1.getBhc().getBurst_duration()*10000);
}
}
else
{
start =n1.getBhc().getOffset_time();
end =  start + n1.getBhc().getBurst_duration();
found = scheduleChannel(start, end,periority);
if (found){
burst_schedule++;
//schedule_burst.add(n1.getBhc().getBhc_id());
		normal_bursts_successful++;
		bitspassed += (n1.getBhc().getBurst_duration()*10000);
if (end > last_clock)
		last_clock = end;
}
else{
                                                                        //unschedule_burst.add(n1.getBhc().getBhc_id());
		normal_bursts_dropped++;
		burst_dropped++;
		bitsdropped += (n1.getBhc().getBurst_duration()*10000);

                                }
                        }
                        eventlist.remove(count);
                        size = eventlist.size();

		}
double totalBurst = burst_schedule+burst_dropped;
indnormalizedthroughput = burst_schedule / totalBurst;
indcont = continuous_bursts_dropped / (continuous_bursts_dropped+continuous_bursts_successful);
                                indper = periodic_bursts_dropped / (periodic_bursts_dropped+periodic_bursts_successful);
                                indnorm = normal_bursts_dropped / (normal_bursts_dropped+normal_bursts_successful);
		for (int i=0;i<wv;i++)
                        {
                                total_utilization += wv_utilization[i];
                        }
                        total_utilization  = (total_utilization / (wv));

                        double bandwidth_util = total_utilization / simulation_time;
                        double lossratio =  (burst_dropped / totalBurst)  ;
                        indloss = lossratio;
                        totalbursts += burst_dropped+burst_schedule;

                        bitspassed = bitspassed / (1000 * 1000 *1000);
```

```
                    simulation_time = simulation_time / (1000 * 1000);

indthroughput = ((bitspassed / simulation_time) )/wv;//  G bits per second
                    indband = (indthroughput / 10) ;


double meaninterarrivaltimeobs = totalinterarrivaltime/(normal_bursts_dropped + normal_bursts_successful);

double meanintarrivaltimecont = totalintarrivalcont / (continuous_bursts_successful+continuous_bursts_dropped) ;
double meanintarrivaltimeperiodic = totalintarrivalper / periodic_bursts_dropped+periodic_bursts_successful);
double mean_burst_length_per = totalperburstsizes / (periodic_bursts_dropped+periodic_bursts_successful);
double mean_burst_length = mean_burst_size/10000;
double mean_burst_length_cont = (mean_burst_size_continuous/cont_burst)/10000;
indobsload = ((1/meaninterarrivaltimeobs) / (1/ mean_burst_length))/wv;
indperload = ((1/meanintarrivaltimeperiodic) / (1/ mean_burst_length_per))/wv;
indcontload = ((1/meanintarrivaltimecont) / (1/ mean_burst_length_cont))/wv;
indload = ((indobsload + indcontload + indperload));
}catch(Exception e){
        System.out.println("Exception"+e.getMessage()+e.getLocalizedMessage());
        }
}
public boolean scheduleChannel(double starttime, double endtime,double periority){
                    int channel=0;
                    boolean schedule = false;
                    boolean scheduleusingvoid = false;
                    boolean scheduleusinglauc = false;

        try{
        double[] startingvoid = new double[wv];
        double[] endingvoid = new double[wv];
        for (int i=0;i<wv;i++){
        if (channel_list.get(i).size() > 0){
                    double startingdiff = 0;
                    double endingdiff = 0;
                    for (int j=0;j<channel_list.get(i).size();j++){
                    if(starttime > Double.parseDouble(channel_list.get(i).get(j).get(2).toString())){
                    if (Double.parseDouble(channel_list.get(i).get(j).get(2).toString())>startingdiff){
                                startingdiff = Double.parseDouble(channel_list.get(i).get(j).get(2).toString());
                                startingvoid[i] = starttime-Double.parseDouble(channel_list.get(i).get(j).get(2).toString());
                    }
                    }
                    }
                    double temp = 999999999999.0;
        for (int j=0;j<channel_list.get(i).size();j++){
        if(Double.parseDouble(channel_list.get(i).get(j).get(1).toString())>startingdiff){
                    if(Double.parseDouble(channel_list.get(i).get(j).get(1).toString())<temp){
                    temp = Double.parseDouble(channel_list.get(i).get(j).get(1).toString());
                    }
        }
        if(temp==999999999999.0)
        {
        }else{
        endingvoid[i]=temp-endtime;
        }
        }
}
}
double minvoid = startingvoid[0];
boolean foundvoid=false;
for (int i=0;i<wv;i++){
        if(startingvoid[i]<=minvoid && endingvoid[i]>0.0  && startingvoid[i]>0.0){
        minvoid = startingvoid[i];
        foundvoid=true;
        channel = i;
        schedule=true;
        }
```

```
}
if(foundvoid){
		channel_list.get(channel).add(channel_list.get(channel).size(),new ArrayList());
		channel_list.get(channel).get(channel_list.get(channel).size()-1).add(periority);
		channel_list.get(channel).get(channel_list.get(channel).size()-1).add(starttime);
		channel_list.get(channel).get(channel_list.get(channel).size()-1).add(endtime);
		wv_utilization[channel]+=(endtime-starttime);
}else{
		boolean foundusinglauc=false;
		double[] laucend = new double[wv];
		for (int i=0;i<wv;i++){
		if (channel_list.get(i).size() > 0){
			double maxlauc =0;
		for (int j=0;j<channel_list.get(i).size();j++){
		if(Double.parseDouble(channel_list.get(i).get(j).get(2).toString())>maxlauc )
		maxlauc=Double.parseDouble(channel_list.get(i).get(j).get(2).toString());
		}
		laucend[i]=maxlauc;
		}
}
double maxlauc =0;
for (int i=0;i<wv;i++){
if(laucend[i]>=maxlauc  && laucend[i]<starttime ){
		maxlauc = laucend[i];
		foundusinglauc=true;
		channel = i;
}
}
if(foundusinglauc){
		schedule=true;
		channel_list.get(channel).add(channel_list.get(channel).size(),new ArrayList());
		channel_list.get(channel).get(channel_list.get(channel).size()-1).add(periority);
		channel_list.get(channel).get(channel_list.get(channel).size()-1).add(starttime);
		channel_list.get(channel).get(channel_list.get(channel).size()-1).add(endtime);
		wv_utilization[channel]+=(endtime-starttime);
}
else{
for (int i=0;i<wv;i++){
		if (channel_list.get(i).size()==0){
		channel = i;
		channel_list.get(channel).add(0,new ArrayList());
		channel_list.get(channel).get(0).add(periority);
		channel_list.get(channel).get(0).add(starttime);
		channel_list.get(channel).get(0).add(endtime);
		schedule = true;
		wv_utilization[channel]+=(endtime-starttime);
		//System.out.println("Channel : "+channel+"  "+ channel_list.get(channel));
		break;
}

}
}
}
if(periority==2 && schedule==false){
		//		System.out.println("testing");
		int[] channel_for_periority = new int[wv];
		int[] check_for_class2 = new int[wv];
		int[] check_for_class1 = new int[wv];
		int[] check_for_class0 = new int[wv];
		double[] size_ofbursts_class0 = new double[wv];
		double[] size_ofbursts_class1 = new double[wv];
		ArrayList<ArrayList> channel_to_drop_burst = new ArrayList<ArrayList>();
		channel_to_drop_burst.clear();
		for (int i=0;i<wv;i++){
		channel_for_periority[i]=0;
		check_for_class2[i]=0;
```

```
check_for_class1[i]=0;
check_for_class0[i]=0;
size_ofbursts_class1[i]=0;
size_ofbursts_class0[i]=0;
channel_to_drop_burst.add(i,new ArrayList<ArrayList>());
//            System.out.println("Size"+channel_list.get(i).size());
for (int j=0;j<channel_list.get(i).size();j++){
if ((Double.parseDouble(channel_list.get(i).get(j).get(1).toString())>starttime  &&
Double.parseDouble(channel_list.get(i).get(j).get(1).toString())<endtime)   ||
(Double.parseDouble(channel_list.get(i).get(j).get(2).toString())<endtime &&
Double.parseDouble(channel_list.get(i).get(j).get(2).toString())>starttime) ){
channel_for_periority[i]++;
channel_to_drop_burst.get(i).add(j);
double sizebur = (Double.parseDouble(channel_list.get(i).get(j).get(1).toString())) -
(Double.parseDouble(channel_list.get(i).get(j).get(1).toString()));
//System.out.println("geo "+j);
if(Double.parseDouble(channel_list.get(i).get(j).get(0).toString())==2)
{
check_for_class2[i]++;
}
else
if(Double.parseDouble(channel_list.get(i).get(j).get(0).toString())==1)
{
check_for_class1[i]++;
size_ofbursts_class1[i] += sizebur;
}
else{
check_for_class0[i]++;
size_ofbursts_class0[i] += sizebur;
}
}
}
}
double minp = 9999999999999.0 ;
int chnl=0;
boolean foundp=false;
for ( int k=0; k<wv; k++ )
{
if ( channel_for_periority[k] <= minp && check_for_class2[k]==0 && check_for_class1[k]==0 && channel_for_periority[k]!=0 ){
minp = channel_for_periority[k];
}
}
double mins = 9999999999999.0 ;
for ( int k=0; k<wv; k++ )
{
if ( channel_for_periority[k] == minp && check_for_class2[k]==0 && check_for_class1[k]==0 && channel_for_periority[k]!=0 ){
if(size_ofbursts_class0[k]<= mins){
            mins = size_ofbursts_class0[k];
            chnl = k;
            foundp = true;
}
//            System.out.println("Chnl 2nd"+chnl);
}
}
if (foundp==false){
minp = 9999999999999.0 ;
for ( int k=0; k<wv; k++ )
{
if ( channel_for_periority[k] <= minp && check_for_class2[k]==0 && channel_for_periority[k]!=0 ){
            minp = channel_for_periority[k];
}
}
mins = 9999999999999.0 ;
for ( int k=0; k<wv; k++ )
{
if ( channel_for_periority[k] == minp && check_for_class2[k]==0  && channel_for_periority[k]!=0 ){
```

```
		if(size_ofbursts_class1[k]<= mins){
			mins = size_ofbursts_class0[k];
			chnl = k;
			foundp = true;
		}
		//		System.out.println("Chnl 1st"+chnl);
		}
		}
		}
		if(foundp){
		schedule=true;
		burst_dropped+= channel_to_drop_burst.get(chnl).size();
		burst_schedule -= channel_to_drop_burst.get(chnl).size();
		for (int l =0 ; l< channel_to_drop_burst.get(chnl).size();l++){
			channel_list.get(chnl).remove(channel_to_drop_burst.get(chnl).get(l));
			int ind = Integer.parseInt(channel_to_drop_burst.get(chnl).get(l).toString());
			double strt = Double.parseDouble(channel_list.get(chnl).get(ind).get(1).toString());
			double ent = Double.parseDouble(channel_list.get(chnl).get(ind).get(2).toString());
			wv_utilization[chnl]-= (ent-strt);
			bitsdropped += (ent-strt);
		}
		channel_list.get(chnl).add(channel_list.get(chnl).size(),new ArrayList());
		channel_list.get(chnl).get(channel_list.get(chnl).size()-1).add(periority);
		channel_list.get(chnl).get(channel_list.get(chnl).size()-1).add(starttime);
		channel_list.get(chnl).get(channel_list.get(chnl).size()-1).add(endtime);
		wv_utilization[chnl]+=(endtime-starttime);
		bitspassed += (endtime-starttime);
		}
		//		System.out.println("Schedule:  "+ schedule);
		}

		}catch(Exception e ){
		System.out.println(e.getMessage());
		}
		return schedule;
}
public  double calculate_arrival_time(double mean){
		//return Math.round(-Math.log(1 - Math.random()) / mean);
		double random=r.nextDouble();
		double exp_time=(-1/mean)*Math.log(random);
		return Math.round(exp_time);
}
public  double calculate_burst_size(double mean){
		double random=r1.nextDouble();
		double exp_time=(-mean)*Math.log(random);
		return Math.round(exp_time);
}
public static void main(String agrs[])
{
		FTMWithQos f = new FTMWithQos();
		double mean_arrival_rate = 0.043;
		for (double m = mean_arrival_rate; mean_arrival_rate<=0.09; mean_arrival_rate+=0.005){
		int samplesize = 20;
		 double[] loss = new double[samplesize];
		     double[] band = new double[samplesize];
		     double[] load = new double[samplesize];
		     double[] throughput = new double[samplesize];
		     double[] northroughput = new double[samplesize];
		     double[] contloss = new double[samplesize];
		     double[] perloss = new double[samplesize];
		     double[] norloss = new double[samplesize];
		double losssum=0,bandsum=0,loadsum=0, throughputsum=0, northroughputsum=0, contsum=0,persum=0,norsum=0;
		try {
		   FileWriter fstream = new FileWriter("qos.txt",true);
		   BufferedWriter out = new BufferedWriter(fstream);
		  for(int i=0; i<samplesize;i++){
```

```
            f.runSimulation(mean_arrival_rate);
              load[i]=indload;
              loadsum+=load[i];
              loss[i]=indloss;
              losssum+=loss[i];
              band[i]=indband;
              bandsum+=band[i];
              throughput[i]=indthroughput;
              throughputsum+=throughput[i];
              northroughput[i]=indnormalizedthroughput;
              northroughputsum+=northroughput[i];
              contloss [i] = indcont;
              contsum += contloss[i];
              perloss [i] = indper;
              persum += perloss[i];
              norloss [i] = indnorm;
              norsum += norloss[i];
 }
double meanloss = losssum / samplesize;
double meanband= bandsum / samplesize;
double meanload= loadsum / samplesize;
double meanthrouput= throughputsum / samplesize;
double meannorthrouput= northroughputsum / samplesize;

double meancontloss = contsum / samplesize;
double meanperloss = persum / samplesize;
double meannorloss = norsum / samplesize;
double xxmeansum = 0.0,xxmeanband=0.0, xxmeanthroughput=0.0,xxmeannorthroughput=0, xxmeancontloss =0,
xxmeanperloss=0,xxmeannorloss=0;
for (int j = 0; j < samplesize; j++) {
xxmeansum += (loss[j] - meanloss) * (loss[j] - meanloss);
xxmeanband += (band[j] - meanband) * (band[j] - meanband);
xxmeanthroughput += (throughput[j] - meanthrouput) * (throughput[j] - meanthrouput);
xxmeannorthroughput += (northroughput[j] - meannorthrouput) * (northroughput[j] - meannorthrouput);
xxmeancontloss += (contloss[j] - meancontloss) * (contloss[j] - meancontloss);
xxmeanperloss += (perloss[j] - meanperloss) * (perloss[j] - meanperloss);
xxmeannorloss += (norloss[j] - meannorloss) * (norloss[j] - meannorloss);
    }
double varianceloss = xxmeansum / (samplesize - 1);
double varianceband = xxmeanband / (samplesize - 1);
double variancethroughput = xxmeanthroughput / (samplesize - 1);
double variancenorthroughput = xxmeannorthroughput / (samplesize - 1);
double variancecontloss = xxmeancontloss / (samplesize -1);
double varianceperloss = xxmeanperloss / (samplesize -1);
double variancenorloss = xxmeannorloss / (samplesize -1);
double stddevloss = Math.sqrt(varianceloss);
double stddevband = Math.sqrt(varianceband);
double stddevthroughput = Math.sqrt(variancethroughput);
double stddevnorthroughput = Math.sqrt(variancenorthroughput);
double stddevcontloss = Math.sqrt(variancecontloss);
double stddevperloss = Math.sqrt(varianceperloss);
double stddevnorloss = Math.sqrt(variancenorloss);
double loloss = meanloss - (1.96 * stddevloss);
double hiloss = meanloss + (1.96 * stddevloss);
double loband = meanband - (1.96 * stddevband);
double hiband = meanband + (1.96 * stddevband);
double lothroughput = meanthrouput - (1.96 * stddevthroughput);
double hithroughput = meanthrouput + (1.96 * stddevthroughput);
double lonorthroughput = meannorthrouput - (1.96 * stddevnorthroughput);
double hinorthroughput = meannorthrouput + (1.96 * stddevnorthroughput);
double locontloss = meancontloss - (1.96 * stddevcontloss);
double hicontloss = meancontloss + (1.96 * stddevcontloss);
double loperloss = meanperloss - (1.96 * stddevperloss);
double hiperloss = meanperloss + (1.96 * stddevperloss);
double lonorloss = meannorloss - (1.96 * stddevnorloss);
double hinorloss = meannorloss + (1.96 * stddevnorloss);
```

```
   out.newLine();
   out.write(Double.toString(meanload));
   out.write("            ");
   out.write(Double.toString(meanloss));
   out.write("            ");
  out.write(Double.toString((hiloss-meanloss)));
   out.write("            ");
   out.write(Double.toString(meanband));
  out.write("            ");
  out.write(Double.toString((hiband-meanband)));
  out.write("            ");
  out.write(Double.toString(meannorthrouput));
  out.write("            ");
  out.write(Double.toString((hinorthroughput-meannorthrouput)));
  out.write("            ");
  out.write(Double.toString(meancontloss));
  out.write("            ");
  out.write(Double.toString((hicontloss-meancontloss)));
  out.write("            ");
  out.write(Double.toString(meanperloss));
  out.write("            ");
  out.write(Double.toString((hiperloss-meanperloss)));
  out.write("            ");
  out.write(Double.toString(meannorloss));
  out.write("            ");
  out.write(Double.toString((hinorloss-meannorloss)));
  out.close();

}catch(Exception e){
            System.out.println("Geo "+ e.getLocalizedMessage());
            }
                        finally {
                        //  write.close();
                        }


                        }

            }

}
// this class is used for simulation FTM Network Without QoS Provisioning
package FTMSimulation;
import java.util.*;
import java.text.DecimalFormat;
import java.io.*;
import FTMSimulation.*;
/*
@Author Muhammad Imran

*/
public class FTMSimulation{
public static double
totalbursts=0,meanbandwidth=0,meanlossrate=0,totalinterarrivaltime=0,totalburstssizes=0,meanloadinper=0,inter=0,mean_length=0;
public static double indload=0,indband=0,indloss=0,indthroughput=0,bitspassed=0,bitsdropped=0, indnormalizedthroughput=0, indcont
=0,indper=0,indnorm=0;
public static double indobsload=0,indcontload=0,indperload=0,totalintarrivalcont=0,totalintarrivalper=0, totalperburstsizes=0;
int controlchannels=1;
double datarate = 10000000000.00;
int nodes = 4;
int wv=6;                   // Total number of wavelength per node
public static Random r;// Random number generator
public static Random r1; // Random number generator
public static Random r2; // Random number generator
double mean_burst_size = 50*1000*8;
double mean_burst_size_continuous = 500*1000*8;
```

```
double[] utilization = new double [wv]; // Bandwidth utilization
double[] wv_utilization = new double [wv]; // Bandwidth used per node
ArrayList<ArrayList<ArrayList>> channel_list = new ArrayList<ArrayList<ArrayList>>();
int burst_schedule = 0;
int burst_dropped =0;
double continuous_bursts_dropped=0,continuous_bursts_successful=0;;
double periodic_bursts_dropped=0,periodic_bursts_successful=0;
double normal_bursts_dropped=0,normal_bursts_successful=0;
public void runSimulation(double mean_arrival_rate){
try{
r=new Random(); // Random number generator
r1=new Random(); // Random number generator
r2=new Random(); // Random number generator
double simulation_time = 500000;
totalbursts=0;meanbandwidth=0;meanlossrate=0;totalinterarrivaltime=0;
meanloadinper=0;inter=0;mean_length=0;
burst_schedule = 0;totalburstssizes=0;
burst_dropped =0;
indthroughput=0;bitspassed=0;bitsdropped=0;
continuous_bursts_dropped=0;continuous_bursts_successful=0;;
periodic_bursts_dropped=0;periodic_bursts_successful=0;
normal_bursts_dropped=0;normal_bursts_successful=0;
indcont =0;indper=0;indnorm=0;
indobsload=0;indcontload=0;indperload=0;totalintarrivalcont=0;
indload=0;indband=0;indloss=0;indthroughput=0;bitspassed=0;bitsdropped=0; indnormalizedthroughput=0; indcont =0;indper=0;indnorm=0;

indobsload=0;indcontload=0;indperload=0;totalintarrivalcont=0;totalintarrivalper=0; totalperburstsizes=0;
            indnormalizedthroughput=0;
            totalperburstsizes=0;
            indload=0;indband=0;indloss=0;
            totalintarrivalper=0;
for (int x=0; x<wv; x++)
{
wv_utilization[x] = 0.0;
utilization[x] = 0.0;
}
channel_list.clear();
for (int i=0;i<wv;i++)
{
channel_list.add(i,new ArrayList<ArrayList>());
}
LinkedList eventlist=new LinkedList(); //create event list
int counter=0;
node n1=null;
int MCL = 0;
Comp1 comp = new Comp1();
Collections.sort(eventlist,comp);
int propogaration_delay = 1000;
int configuration_delay = 3;
int cont_burst=10;
int periodic_burst = 10;
int cp_id=1;
int dest = 3;
double total_utilization=0;
int mode =0;
//Obs mode generation
while (counter<simulation_time){
double inter_arrival_time = calculate_arrival_time(mean_arrival_rate);
MCL += inter_arrival_time;
totalinterarrivaltime += inter_arrival_time;
double burst_size = calculate_burst_size(mean_burst_size);
double burst_duration = Math.round((burst_size)/10000);
CP cp;
//          totalburstssizes += burst_duration;
cp = new CP(0,          0, dest, propogaration_delay,configuration_delay, MCL,
MCL+propogaration_delay,MCL+propogaration_delay+inter_arrival_time, cp_id+"p",burst_duration);
```

```
counter+=inter_arrival_time;
node n2 = new node(2,cp.getArrival_time(), 1, cp);
n2.setBhc(cp);
eventlist.add(n2);
cp_id++;
}

// Continuous Stream Generation
MCL = 0;
counter =0;
while (counter<simulation_time){
double inter_arrival_time = calculate_arrival_time(mean_arrival_rate);
            inter_arrival_time = inter_arrival_time * cont_burst;
            MCL += inter_arrival_time;
            totalintarrivalcont += inter_arrival_time;
            double burst_size = calculate_burst_size(mean_burst_size_continuous);
            double burst_duration = Math.round((burst_size)/10000);
            CP cp;
cp = new CP(2,2, dest, propogaration_delay,configuration_delay, MCL, MCL+propogaration_delay,
MCL+propogaration_delay+inter_arrival_time , cp_id+"p",burst_duration);
            counter+= inter_arrival_time;
            node n2 = new node(2,cp.getArrival_time(), 1, cp);
            n2.setBhc(cp);
            eventlist.add(n2);
            cp_id++;
}

// Periodic Stream Generation

MCL = 0;
counter =0;
while (counter<simulation_time){
double inter_arrival_time = calculate_arrival_time(mean_arrival_rate);
inter_arrival_time = inter_arrival_time;
MCL += inter_arrival_time;
totalintarrivalper += inter_arrival_time;
double burst_size = calculate_burst_size(mean_burst_size);
double burst_duration = Math.round((burst_size)/10000);
totalperburstsizes += burst_duration*periodic_burst;
CP cp;
cp = new CP(1,          1, dest, propogaration_delay,configuration_delay, MCL,
MCL+propogaration_delay,MCL+propogaration_delay+(inter_arrival_time), cp_id+"p",burst_duration,inter_arrival_time);
totalintarrivalper += inter_arrival_time*(periodic_burst-1);
counter+= inter_arrival_time*(periodic_burst-1);
MCL+= inter_arrival_time*(periodic_burst-1);
node n2 = new node(2,cp.getArrival_time(), 1, cp);
n2.setBhc(cp);
eventlist.add(n2);
cp_id++;
}
Collections.sort(eventlist,comp);
int size = eventlist.size();
n1= (node) eventlist.get(0);
int count = 0;
double master_clock=0, last_clock=0;
double initialclock = n1.getBhc().getOffset_time();
while (count<size)
{
Collections.sort(eventlist,comp);
n1= (node) eventlist.get(count);
master_clock= n1.getClock();
double start = 0;
double end = 0 ;
double periority =  n1.getBhc().getPeriority();
boolean found =false;
if (periority==1) {
```

```
start = n1.getBhc().getOffset_time();
end = start + n1.getBhc().getBurst_duration();
for(int i =0; i<periodic_burst ; i++){
        found = scheduleChannel(start, end,periority);
if(found){
        burst_schedule++;
        periodic_bursts_successful++;
        bitspassed += (n1.getBhc().getBurst_duration()*10000);
        if(end > last_clock)
        last_clock = end;
        }
        else{
        burst_dropped++;
        periodic_bursts_dropped++;
        bitsdropped += (n1.getBhc().getBurst_duration()*10000);
        }
start = end + n1.getBhc().getInterarrivaltime();
end = start + n1.getBhc().getBurst_duration();

}
}
else
if (periority==2) {
        start = n1.getBhc().getOffset_time();
        end = start + n1.getBhc().getBurst_duration();
        found = scheduleChannel(start, end,periority);
if(found){
burst_schedule +=cont_burst;
continuous_bursts_successful +=cont_burst;
bitspassed += (n1.getBhc().getBurst_duration()*10000);
if (end>last_clock) {
        last_clock = end;
}
}
else{
burst_dropped+=cont_burst;
continuous_bursts_dropped+=cont_burst;
bitsdropped += (n1.getBhc().getBurst_duration()*10000);
}
}
else
{
start =n1.getBhc().getOffset_time();
end =  start + n1.getBhc().getBurst_duration();
found = scheduleChannel(start, end,periority);
if (found){
burst_schedule++;
///schedule_burst.add(n1.getBhc().getBhc_id());
        normal_bursts_successful++;
        bitspassed += (n1.getBhc().getBurst_duration()*10000);

        if (end > last_clock)
        last_clock = end;
        }
        else{
        normal_bursts_dropped++;
        burst_dropped++;
        bitsdropped += (n1.getBhc().getBurst_duration()*10000);
                }
        }
        eventlist.remove(count);
        size = eventlist.size();
        }
        double totalBurst = burst_schedule+burst_dropped;
        indnormalizedthroughput = burst_schedule / totalBurst;
        indcont = continuous_bursts_dropped / (continuous_bursts_dropped+continuous_bursts_successful);
```

```
		indper = periodic_bursts_dropped / (periodic_bursts_dropped+periodic_bursts_successful);
		indnorm = normal_bursts_dropped / (normal_bursts_dropped+normal_bursts_successful);
		for (int i=0;i<wv;i++)
		{
		total_utilization += wv_utilization[i];
		}
		total_utilization  = (total_utilization / (wv));
		double bandwidth_util = total_utilization / simulation_time;
		double lossratio =  (burst_dropped / totalBurst)  ;
		indloss = lossratio;
		totalbursts += burst_dropped+burst_schedule;
		bitspassed = bitspassed / (1000 * 1000 *1000);
		simulation_time = simulation_time / (1000 * 1000);
		indthroughput = ((bitspassed / simulation_time) )/wv;//  G bits per second
		indband = (indthroughput / 10) ;
		double meaninterarrivaltimeobs = totalinterarrivaltime/(normal_bursts_dropped + normal_bursts_successful);
		double meanintarrivaltimecont = totalintarrivalcont / (continuous_bursts_successful+continuous_bursts_dropped) ;
		double meanintarrivaltimeperiodic = totalintarrivalper / (periodic_bursts_dropped+periodic_bursts_successful);
		double mean_burst_length_per = totalperburstsizes / (periodic_bursts_dropped+periodic_bursts_successful);
		double mean_burst_length = mean_burst_size/10000;
		double mean_burst_length_cont = (mean_burst_size_continuous/cont_burst)/10000;
		indobsload = ((1/meaninterarrivaltimeobs) / (1/ mean_burst_length))/wv;
		indperload = ((1/meanintarrivaltimeperiodic) / (1/ mean_burst_length_per))/wv;
		indcontload = ((1/meanintarrivaltimecont) / (1/ mean_burst_length_cont))/wv;
		indload = ((indobsload + indcontload + indperload));
}catch(Exception e){
System.out.println("Exception"+e.getMessage()+e.getLocalizedMessage());
			}
		}
public boolean scheduleChannel(double starttime, double endtime,double periority){
		int channel=0;
		boolean schedule = false;
		boolean scheduleusingvoid = false;
		boolean scheduleusinglauc = false;
		try{
		double[] startingvoid = new double[wv];
		double[] endingvoid = new double[wv];
		for (int i=0;i<wv;i++){
			if (channel_list.get(i).size() > 0){
			double startingdiff = 0;
			double endingdiff = 0;
			for (int j=0;j<channel_list.get(i).size();j++){
			if(starttime > Double.parseDouble(channel_list.get(i).get(j).get(2).toString())){
			if (Double.parseDouble(channel_list.get(i).get(j).get(2).toString())>startingdiff){
			startingdiff = Double.parseDouble(channel_list.get(i).get(j).get(2).toString());
			startingvoid[i] = starttime-Double.parseDouble(channel_list.get(i).get(j).get(2).toString());
			}
			}
			}
			double temp = 999999999999.0;
			for (int j=0;j<channel_list.get(i).size();j++){
			if(Double.parseDouble(channel_list.get(i).get(j).get(1).toString())>startingdiff){
			if(Double.parseDouble(channel_list.get(i).get(j).get(1).toString())<temp){
			temp = Double.parseDouble(channel_list.get(i).get(j).get(1).toString());
			}
			}
			if(temp==999999999999.0)
			{
			}else{
			endingvoid[i]=temp-endtime;
			}
			}
			}
			}
			double minvoid = startingvoid[0];
			boolean foundvoid=false;
```

```
                for (int i=0;i<wv;i++){
                if(startingvoid[i]<=minvoid && endingvoid[i]>0.0  && startingvoid[i]>0.0){
                minvoid = startingvoid[i];
                foundvoid=true;
                channel = i;
                schedule=true;
                }
                }
                if(foundvoid){
                        channel_list.get(channel).add(channel_list.get(channel).size(),new ArrayList());
                        channel_list.get(channel).get(channel_list.get(channel).size()-1).add(periority);
                        channel_list.get(channel).get(channel_list.get(channel).size()-1).add(starttime);
                        channel_list.get(channel).get(channel_list.get(channel).size()-1).add(endtime);
                        wv_utilization[channel]+=(endtime-starttime);
                }else{
                        boolean foundusinglauc=false;
                        double[] laucend = new double[wv];
                        for (int i=0;i<wv;i++){
                        if (channel_list.get(i).size() > 0){
                        double maxlauc =0;
                        for (int j=0;j<channel_list.get(i).size();j++){
                        if(Double.parseDouble(channel_list.get(i).get(j).get(2).toString())>maxlauc )
                        maxlauc=Double.parseDouble(channel_list.get(i).get(j).get(2).toString());
                        }
                        laucend[i]=maxlauc;
                        }
                }
                double maxlauc =0;
                for (int i=0;i<wv;i++){
                if(laucend[i]>=maxlauc  && laucend[i]<starttime ){
                        maxlauc = laucend[i];
                        foundusinglauc=true;
                        channel = i;
                        }
                }
                if(foundusinglauc){
                schedule=true;
                channel_list.get(channel).add(channel_list.get(channel).size(),new ArrayList());
                channel_list.get(channel).get(channel_list.get(channel).size()-1).add(periority);
                channel_list.get(channel).get(channel_list.get(channel).size()-1).add(starttime);
                channel_list.get(channel).get(channel_list.get(channel).size()-1).add(endtime);
                wv_utilization[channel]+=(endtime-starttime);
                }
                else{
                for (int i=0;i<wv;i++){
                if (channel_list.get(i).size()==0){
                        channel = i;
                        channel_list.get(channel).add(0,new ArrayList());
                        channel_list.get(channel).get(0).add(periority);
                        channel_list.get(channel).get(0).add(starttime);
                        channel_list.get(channel).get(0).add(endtime);
                        schedule = true;
                        wv_utilization[channel]+=(endtime-starttime);
                        //System.out.println("Channel : "+channel+"  "+ channel_list.get(channel));
                break;
                }
        }
        }
        }
        }catch(Exception e ){
                System.out.println(e.getMessage());
        }
        return schedule;
}

        public  double calculate_arrival_time(double mean){
```

```
                double random=r.nextDouble();
                double exp_time=(-1/mean)*Math.log(random);
                return Math.round(exp_time);
}

public  double calculate_burst_size(double mean){

                double random=r1.nextDouble();
                double exp_time=(-mean)*Math.log(random);
                return Math.round(exp_time);
                }
public static void main(String agrs[])
{
                FTMSimulation f = new FTMSimulation();
                double mean_arrival_rate = 0.043;
                for (double m = mean_arrival_rate; mean_arrival_rate<=0.09; mean_arrival_rate+=0.005){
                int samplesize = 20;

      double[] loss = new double[samplesize];
      double[] band = new double[samplesize];
      double[] load = new double[samplesize];
      double[] throughput = new double[samplesize];
      double[] northroughput = new double[samplesize];
      double[] contloss = new double[samplesize];

      double[] perloss = new double[samplesize];

      double[] norloss = new double[samplesize];

double losssum=0,bandsum=0,loadsum=0, throughputsum=0, northroughputsum=0, contsum=0,persum=0,norsum=0;
try {
   FileWriter fstream = new FileWriter("eqhmbftm.txt",true);
   BufferedWriter out = new BufferedWriter(fstream);
                 for(int i=0; i<samplesize;i++){
                 f.runSimulation(mean_arrival_rate);
                 load[i]=indload;
                 loadsum+=load[i];
                 loss[i]=indloss;
                 losssum+=loss[i];
                 band[i]=indband;
                 bandsum+=band[i];
                 throughput[i]=indthroughput;
                 throughputsum+=throughput[i];
                 northroughput[i]=indnormalizedthroughput;
                 northroughputsum+=northroughput[i];
                 contloss [i] = indcont;
                 contsum += contloss[i];
                 perloss [i] = indper;
                 persum += perloss[i];
                 norloss [i] = indnorm;
                 norsum += norloss[i];
                }
 double meanloss = losssum / samplesize;
 double meanband= bandsum / samplesize;
 double meanload= loadsum / samplesize;
 double meanthrouput= throughputsum / samplesize;
 double meannorthrouput= northroughputsum / samplesize;
 double meancontloss = contsum / samplesize;
 double meanperloss = persum / samplesize;
 double meannorloss = norsum / samplesize;
 System.out.println("Mean Load "+meanload);
 System.out.println("Mean Band " +meanband);
 System.out.println("Mean Throughput " +meanthrouput);
 System.out.println("Mean Normalized Throughput " +meannorthrouput);
System.out.println("Mean Continous Loss " +meancontloss);
 System.out.println("Mean Periodic Loss " +meanperloss);
```

```
            System.out.println("Mean Normal loss " +meannorloss);
            double xxmeansum = 0.0,xxmeanband=0.0, xxmeanthroughput=0.0,xxmeannorthroughput=0, xxmeancontloss =0,
xxmeanperloss=0,xxmeannorloss=0;
    for (int j = 0; j < samplesize; j++) {
            xxmeansum += (loss[j] - meanloss) * (loss[j] - meanloss);
            xxmeanband += (band[j] - meanband) * (band[j] - meanband);
            xxmeanthroughput += (throughput[j] - meanthrouput) * (throughput[j] - meanthrouput);
            xxmeannorthroughput += (northroughput[j] - meannorthrouput) * (northroughput[j] - meannorthrouput);
            xxmeancontloss += (contloss[j] - meancontloss) * (contloss[j] - meancontloss);
            xxmeanperloss += (perloss[j] - meanperloss) * (perloss[j] - meanperloss);
            xxmeannorloss += (norloss[j] - meannorloss) * (norloss[j] - meannorloss);
     }
     double varianceloss = xxmeansum / (samplesize - 1);
     double varianceband = xxmeanband / (samplesize - 1);
     double variancethroughput = xxmeanthroughput / (samplesize - 1);
     double variancenorthroughput = xxmeannorthroughput / (samplesize - 1);
     double variancecontloss = xxmeancontloss / (samplesize -1);
     double varianceperloss = xxmeanperloss / (samplesize -1);
     double variancenorloss = xxmeannorloss / (samplesize -1);
     double stddevloss = Math.sqrt(varianceloss);
     double stddevband = Math.sqrt(varianceband);
     double stddevthroughput = Math.sqrt(variancethroughput);
     double stddevnorthroughput = Math.sqrt(variancenorthroughput);
     double stddevcontloss = Math.sqrt(variancecontloss);
     double stddevperloss = Math.sqrt(varianceperloss);
     double stddevnorloss = Math.sqrt(variancenorloss);
     double loloss = meanloss - (1.96 * stddevloss);
     double hiloss = meanloss + (1.96 * stddevloss);
     double loband = meanband - (1.96 * stddevband);
     double hiband = meanband + (1.96 * stddevband);
     double lothroughput = meanthrouput - (1.96 * stddevthroughput);
     double hithroughput = meanthrouput + (1.96 * stddevthroughput);
     double lonorthroughput = meannorthrouput - (1.96 * stddevnorthroughput);
     double hinorthroughput = meannorthrouput + (1.96 * stddevnorthroughput);
     double locontloss = meancontloss - (1.96 * stddevcontloss);
     double hicontloss = meancontloss + (1.96 * stddevcontloss);
     double loperloss = meanperloss - (1.96 * stddevperloss);
     double hiperloss = meanperloss + (1.96 * stddevperloss);
     double lonorloss = meannorloss - (1.96 * stddevnorloss);
     double hinorloss = meannorloss + (1.96 * stddevnorloss);
     System.out.println("average     loss     = " + meanloss);
     System.out.println("sample variance loss = " + varianceloss);
     System.out.println("sample stddev loss   = " + stddevloss);
     System.out.println("95% approximate confidence interval");
     System.out.println("[ " + (loloss-meanloss) + ", " + (hiloss-meanloss) + " ]");
     System.out.println("average     band     = " + meanband);
     System.out.println("sample variance band = " + varianceband);
     System.out.println("sample stddev band   = " + stddevband);
     System.out.println("95% approximate confidence interval");
     System.out.println("[ " + (loband-meanband) + ", " + (hiband-meanband) + " ]");
   out.newLine();
   out.write(Double.toString(meanload));
   out.write("            ");
   out.write(Double.toString(meanloss));
   out.write("            ");
   out.write(Double.toString((hiloss-meanloss)));
   out.write("            ");
   out.write(Double.toString(meanband));
   out.write("            ");
   out.write(Double.toString((hiband-meanband)));
//  out.write("          ");
 //  out.write(Double.toString(meanthrouput));
 //  out.write("          ");
//  out.write(Double.toString((hithroughput-meanthrouput)));
   out.write("            ");
   out.write(Double.toString(meannorthrouput));
```

```
   out.write("              ");
   out.write(Double.toString((hinorthroughput-meannorthrouput)));
  out.write("              ");
   out.write(Double.toString(meancontloss));
   out.write("              ");
   out.write(Double.toString((hicontloss-meancontloss)));
   out.write("              ");
  out.write(Double.toString(meanperloss));
  out.write("              ");
   out.write(Double.toString((hiperloss-meanperloss)));
   out.write("              ");
   out.write(Double.toString(meannorloss));
  out.write("              ");
  out.write(Double.toString((hinorloss-meannorloss)));
   out.close();
}catch(Exception e){
              System.out.println("Geo "+ e.getLocalizedMessage());
}
finally {
//  write.close();
}

}

              }

}
class Comp1 implements Comparator {
                public int compare (Object n1, Object n2) {

                  if(((node)n1).getClock() > ((node)n2).getClock())
                              return 1;
                  else if(((node)n1).getClock() == ((node)n2).getClock()) return 0;
                  else return -1;

              }              }
```