Minimizing Wavelength Contention & Improving Packet Level Performance of Time Sliced Optical Burst Switching (TS-OBS) Network

By

Farhan Sabir Ujager

(2009-NUST-MS-Phd-IT-09)

A thesis submitted in partial fulfillment of

the requirements of the degree of

Masters in Information Technology

In

School of Electrical Engineering and Computer Science (SEECS)

National University of Science and Technology (NUST)

Islamabad, Pakistan

Certificate

Certified that the contents and form of thesis entitled "Minimizing wavelength contention and Improving packet level performance of an OBS Network" submitted by Mr. Farhan Sabir Ujager has been found satisfactory for the requirement of degree.

Supervisor: _________________
Dean (Dr. S. M. H. Zaidi)

Member: ___________________
Mr. Muhammad Ramzan

Member: ___________________
Dr. Usman Younas

Member: ___________________
Dr. Tauseef Tauqeer

Dedicated

To Almighty God

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 SEECS or any other education institute, except where due acknowledgement is made in the thesis. Any contribution made to the research by others, with whom I have worked at 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 to the extent that assistance from others in the project is design and conception or in style, presentation and linguistic is acknowledged.

Signature: _______________

Farhan Sabir Ujager

Acknowledgement

With the blessings of Almighty God I have been able to complete my thesis in time. I pray to Almighty God for His mercy and blessings, so as to have a successful life in future.

I am thankful to my supervisor Dr S. M. H. Zaidi for his valuable advices and all out moral support. It was because of his guidance that this research project finally came to an end.

I also pay thanks to my committee members, Dr.Usman Younis, Mr. Muhmmad Ramzan, and Dr.Tauseef Tauqeer for their guidance and support in completion of this thesis.

I am also grateful to my friends and fellow students who supported and helped me through their suggestions and provided good company during the thesis.

Farhan Sabir Ujager

Table of Contents

List of Abbreviations

OBS	Optical Burst Switching
TS-OBS	Time Slotted/Sliced OBS
LASER	Light Amplification through Stimulated Emission of Radiation
LED	Light Emitting Diode
FP LASER	Fabry Perot LASER
EDFA	Erbium Doped Fiber Amplifier
SOA	Semiconductor Optical Amplifier
SRS	Stimulated Raman Scattering
WDM	Wavelength Division Multiplexing
DWDM	Dense Wavelength Division Multiplexing
OCS	Optical Circuit Switching
OPS	Optical Packet Switching
OEO	Optical Electrical Optical
FDL	Fiber Delay Lines
ATM	Asynchronous Transfer Mode
AON	All Optical Network
RAM	Random Access Memory
TAG	Tell And Go
RFD	Reserved for Fix Duration
FEC	Forward Equivalent Class
MPLS	Multi Protocol Label Switching
FFUC	First Fit Unscheduled Channel
LAUC	Latest Available Unscheduled Channel
LAUC-VF	LAUC with Void Filling
G-LAUC-VF	Generalized-LAUC-VF
DB	Data Burst

CP	Control Packet
C-OBS	Conventional OBS
ITU	International Telecommunication Union
OXC	Optical Cross Connect
AFQD	Absolute Fair Quality of service Differentiation scheme
OTWP	Optimization Topology aware Wavelength Partitioning scheme
ILP	Integer Linear Programming
QoS	Quality of Service

List of Tables

List of Figures

Abstract

Optical Burst Switching (OBS) is one of the promising switching technique to meet the demands of the future networks. OBS combines the feature of optical circuit switching (OCS) and optical packet switching (OPS) which highlights its potential to meet the challenges of the future all optical networks (AON). In this work we present a modified network architecture of time sliced optical burst switching (TSOBS), which shows improved performance in terms of lower packet transmission delay. The motivation proposed work is to minimize wavelength contention and improve packet level performance at the edge node of an OBS network. Separate data and synchronization planes are defined for the data burst transmission and synchronization of all core network nodes, respectively. In order to achieve lower transmission delay and minimizing wavelength contention, different time slots are associated with wavelengths for the establishment of light path using modulo arithmetic. Additionally, we present a novel delay aware burst transmission algorithm which is designed to lower data loss probability. The performance of our proposed work and of those proposed in the literature is studied through computer simulation. The performance results indicate that our proposed solution performs significantly better than recently published work in this domain.

1 Optical communication

Optical communication uses optical fiber as a transmission medium, in which data is transmitted as an optical signal. Optical communication needs an optical transmitter to inject data into the optical fiber as light signals, laser diode or light emitting diode (LED) are used to serve this purpose. At the termination end of the fiber there should be a specialized device that receive the data in optical domain and convert that data into electrical domain, photo diodes serve this purpose of receiving data in optical domain and then converting into electrical domain for further transmission to access and metro networks. The concept of optical communication is summarized in the optical communication schematic (figure 1.1).

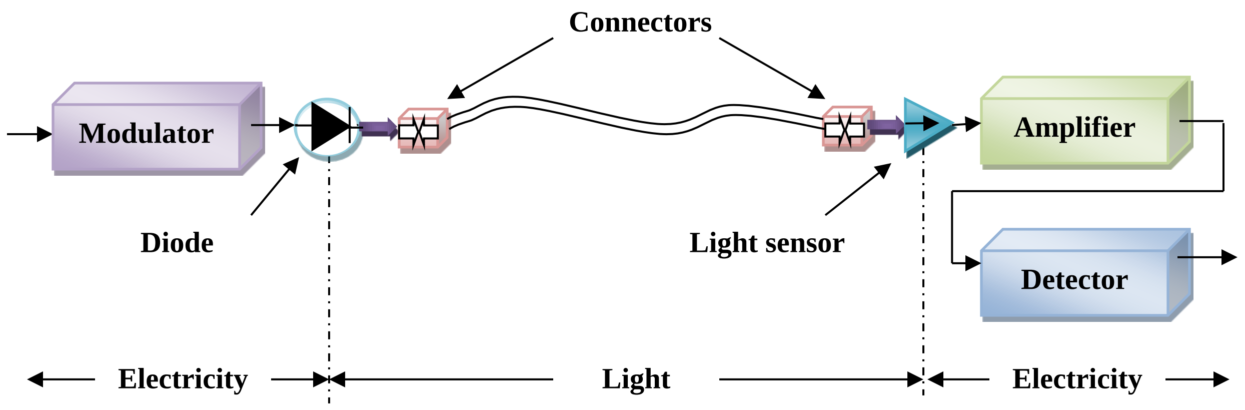


Figure1.1 Optical communication schematic

The basic concept of the optical communication has been shown in the figure1.1

- Data in electrical domain is coming to the modulator in serial bit format and modulator encods the data in the suitable form for the fiber transmission
- Diode is either laser diode or LED which is driven by the modulator and the light is focused into the optical fiber
- During the propagation of light inside the fiber data signal may experience dispersion or loss of strength

- At the receiving end light signals are detected by the detector and converted into proportional electrical signals
- The received signal is amplified and then fed to another detector for the isolation of the states, from here the original bit stream is reconstructed

1.1 Optical Communication Components

1.1.1 Optical transmitter

Optical light source for the optical communication is either LED or solid state laser diode. Both these light sources can be used as an optical transmitter but laser diode produce more coherent light as compare to LED. Infrared light is used rather than visible light for the transmission as it incurs less attenuation and dispersion. The most popular wavelengths which are being used by the optical transmitter are 1330 and 1550 nm.

Laser diode is the most popular optical data transmitter, Semiconductor lasers are compact is size, low in power consumption , produce coherent light and their low cost make them ideal to meet the current and next generation network demands[1].

Optical transmitters must be designed to be reliable, compact and directly modulated at high frequencies [2]. As optical communication is marching to technological advancements these key design parameters are always kept in mind.

In semiconductor laser gain medium is semiconductor. Emission of light can take place due to spontaneous or stimulated emission. Spontaneous emission can takes place when electron from its ground state move to any higher level, as electron is in unstable at the higher energy level stays for a very short time (in ps) and return to its ground state by emitting photon. The direction and phase of the emitter light is random. In stimulated emission of light is emitted when electron is raised to higher level which is the excited state, electron stay there for longer period in μs, before it changes the state spontaneously. When electron enters to meta-stable state, higher than the ground state, electron can be stimulated by the presence of a photon of light to emit its energy in the form of another photon with the same wavelength, phase and direction. Either of the emission to take place need external source of energy for the raising up of electron from its ground state, this external energy can be in the form of electrical or thermal energy.

LED is ideal for short distance communication and low data rate applications as it is low in cost as compare to laser diodes.

LED is basically a p-n junction diode operates in forward biasing. When electric potential is applied across the p-n junction, electrons from the n-region and holes in the p-region are pushed to cross the junction, due to forward biasing. As soon as they cross the junction most of electrons will recombine with holes and eliminate each other. When this happens free electrons will lose their quantum of energy in the form of light to fill the accessible holes. The wavelength of the radiated energy depends on the energy gap the free electron has crossed to fill the hole.

1.1.2 Optical Fiber

Optical fiber is the sole optical signal carrying medium, shown in figure 1.2. It is a thin silica glass strand; its geometry is similar to human hair. Optical fiber is a long glass cylinder and very narrow in nature. When light enters into the fiber it remains confined into the fiber until it leaves the fiber. The journey of light inside the fiber is based on the principle of total internal reflection.

The most important designing features of optical fiber are [3]

- Very small loss of light during its journey into the fiber
- Fiber should be flexible enough for the bends and the light should remain inside the fiber and guided

While injecting light inside the fiber it is very important to know which type of light is injected in terms of wavelength. With the change in the wavelength of light, characteristics of propagation of light also changes, for example, at 1330 nm of wavelength of light the propagated light has minimum dispersion and at 1550 light has minimum attenuation.

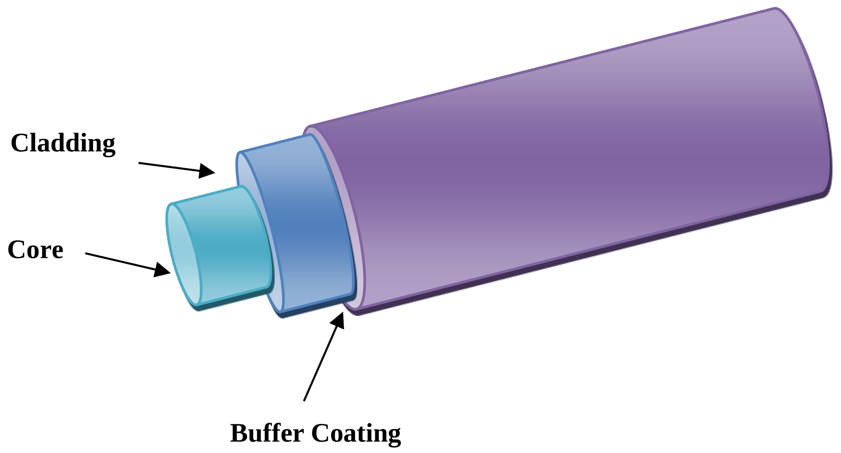


Figure 1.2: Fiber optic cable

1.1.2.1 Principle of propagation

The fiber optic consists of two parts (a) core and (b) cladding, as shown in figure 1.3. The core is a narrow cylinder in which light is injected while cladding is a tube like jacket over the core. The refractive index of the core is higher than of cladding for the reason when light go through the process of total internal reflection remain inside the core.

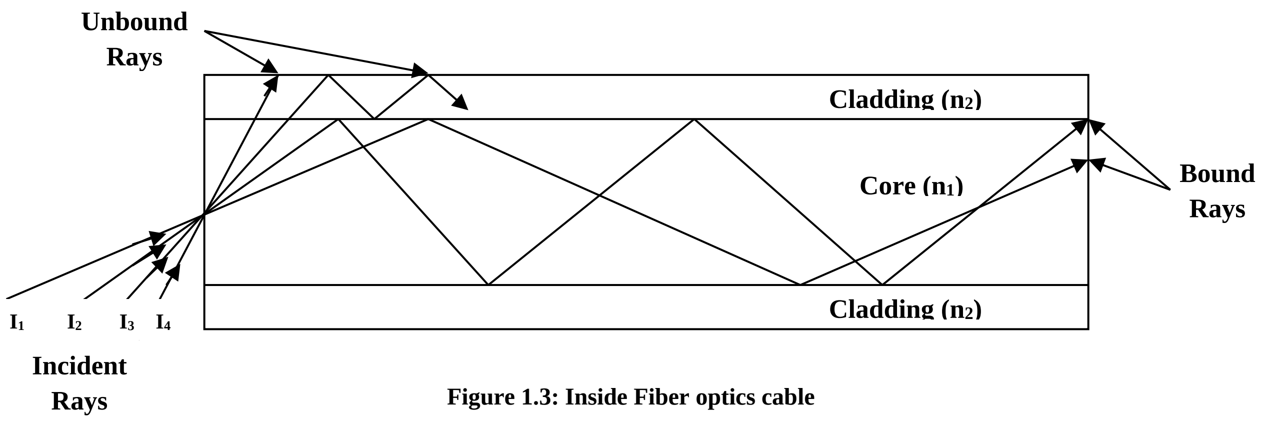


Figure 1.3: Inside Fiber optics cable

There are four incident rays I_1, I_2, I_3 and I_4. The incident rays I_4 and I_3 have the incident angle less than the critical angle which makes them to pass through the core and enter into the cladding making them unbound rays. The other two incident rays I_1 and I_2 have their incident angle values equal or greater than critical angle, which makes them to remain inside the core (bound rays) as shown in the figure 1.3.

1.1.3 Optical receiver

Optical receiver, receive optical signal and convert that signal to a proportional and usable electrical signal. Optical receiver is also known as Photo detector, while discussing photo detector there are some vital parameters which must be kept in mind [3].

- Detector responsivity: This parameter is basically the efficiency of the detector; it is the ratio of the output electrical current to the input optical power.
- Spectral response range: It defines the range of the wavelengths for a device over which the device will operate.
- Response time: It defines that how quickly detector can respond to the variation in the incident optical power.
- Noise characteristics: It defines the noise level at which the input optical signal will be detected accurately.

Different components of the receiver are shown in the figure 1.4.

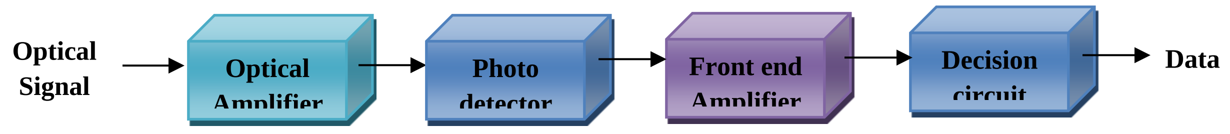


Figure 1.4: Block diagram of receiver in optical communication system

The photo detector generates the proportional electrical current for the input optical power, the front end amplifier amplify electric current to the usable level and the decision circuit works on the amplified electric signals and depends on the modulation technique used.

PIN diodes and avalanche diodes are the two famous types of photo detectors.

1.1.2.1 PIN diode

The working mechanism of the PIN diode is similar to the principle of LED in reverse. When p-n junction is in reverse biased mode, it does not allow current to pass through it until something happens at the depletion region to promote electron from the valence band to the conduction band. When photon of sufficient energy falls it raises the electron from the valence band to the conduction band creating holes and free electrons. This absorption of photon causes the attraction of holes and free electrons to their opposite charges to their sides and current flows across the junction.

1.1.2.2 Avalanche Photo diode

We have just discussed that one photon of sufficient energy is required to create one electron. If this free electron is subjected to high electric field, this free electron will then have sufficient energy to raise other electrons from the valence band to the conduction band, thus creating secondary free electron-hole pairs. These secondary electron-hole pairs will further raise more electrons from valence band to the conduction band when they are accelerated to sufficient level, thus avalanche process has started. This process is called avalanche multiplication and diode is called avalanche photo diode.

1.1.3 Optical Amplifier

Optical amplifier amplifies the optical signals without converting them into electric current. The advent of optical amplifier has revolutionized the modern optical communication. The most popular types of optical amplifiers are erbium doped fiber amplifier (EDFA), raman amplifier and semiconductor optical (SOA). The working principles of all above mentioned amplifiers are different.

1.1.3.1 *EDFA*

It consists of short length fiber whose core is doped with ionized particles (ions), Er^{+3}, a rare earth element erbium. Doping is done in controlled manner to achieve the desired results. It has many advantages, which makes it ideal for modern optical communication [3].

- Availability of reliable and compact high power laser pumps, makes EDFA operation efficient

- Simplicity of the device operation
- No cross talk while amplifying WDM signals

Working principle:

The amplification principle in EDFA is based on the stimulated emission of radiation. Erbium ion has a property that it can exist in multiple energy levels.

High (relatively, for pumping) beam of light is mixed with input signals to pass through that section of fiber in which the core is doped with erbium ions, Er^{3+}. When light passes through this section it excites the ions to higher energy levels and as signal light (photons belonging to the signal) passes through this specific section excited ions give away their energy and come back to their ground state. They give up energy in the form of light (photon) with the same phase and direction to the input signal, usually an isolator is placed at the output to prevent reflection as shown in the figure 1.5. To excite the erbium ions high power pump laser is used, as mentioned, only at specific wavelength of 980 nm.

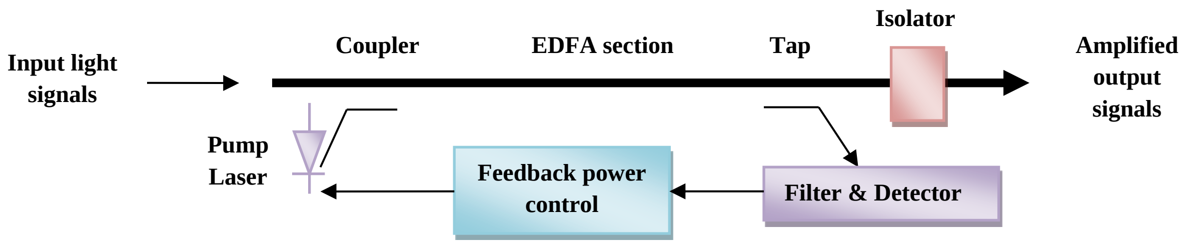


Figure 1.5: EDFA block diagram

1.1.3.2 ***SOA***

SOA uses semiconductor as a gain medium. The working principle is almost same to the Fabry Perot LASERS. SOAs are not good as EDFA, we will discuss their limitations later in this section.

SOA is essentially a p-n junction and light is amplified when it passes through the active region as shown in the figure 1.6.

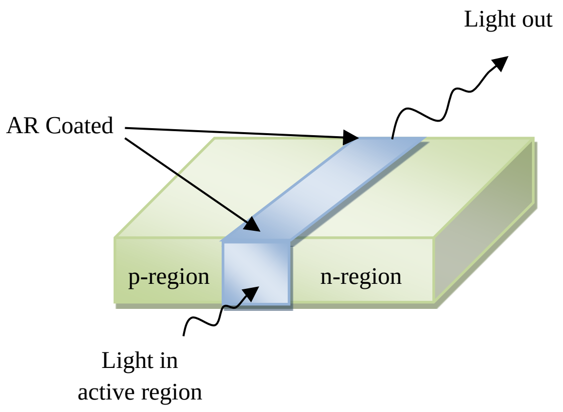


Figure 1.6: Block diagram of SOA

In SOA p-n junction is active region, light is amplified when it is propagated through the active region. As its working principle is similar to the FP LASER diode except in laser diode no antireflection coating is used. Considering p-region, when the number of electrons is more in conduction band than the valence band such situation is called population inversion (similarly if we consider n-region, then for population inversion to occur number of holes are more in conduction band than valence band). Population inversion in SOA is achieved though forward biasing of the p-n junction and input signal is amplified when it propagates through the active region by stimulated emission.

Limitations of SOA:

- SOA is usually suitable for single channel operation as it cannot deliver much power in mW
- SOA tends to be noisy [4]
- They introduce cross talk when multiple channels are amplified

1.1.3.3 ***Raman Effect amplifier:***

Raman Effect amplification is based on stimulated raman scattering (SRS), if two or more signals at different wavelengths are injected into the fiber, SRS causes power transfer from low wavelength channel to the high wavelength, this is the fundamental principle of amplification in Raman Effect amplifier. The transfer of energy from low wavelength channel to the high wavelength channel corresponds to photons of higher energy stimulates the emission of photons on low energy. If we look into the mathematical explanation of photon energy, photon of wavelength λ is expressed as $\boldsymbol{E = hc / \lambda}$**,** where $\boldsymbol{h}$ is the planks constant and $\boldsymbol{c}$ is the speed of light, thus photons of low wavelength have higher energy.

We can use Raman Effect to provide gain at a wavelength; EDFA is restricted to provide gain in the ***C*** and ***L*** bands (1528-1605nm). Other WDM bands are open to provide gain, such as 1310 nm window ***S*** band, because of raman effect amplification.

1.2 Optical Network

Optical networks have very high data carrying capacity. These networks are based on optical technologies and components that are used for switching, routing, grooming, conversion and restoration at the wavelength level and as well as wavelength based services level. Optical networks have very attractive benefits other than high data carrying capacity such as reduced cost at data transmission level, transparency, increased network quality and many more.

1.2.1 OPTICAL CIRCUIT SWITCHING (OCS)

OCS is based on the concept of establishing lightpath from the source to the destination for the transferring of data over a specific available wavelength. Establishment of lightpath (circuit) involves several tasks such as topology description, resource discovery, assigning wavelength, signaling and reserving resources. Once the lightpath is established between specific source and destination this circuit is not available for any other source and destination pair.

It is difficult to create full mesh among all nodes of a network because of limited number of wavelengths are available in a single fiber.

Sending highly variable and bursty nature traffic results in the inefficient utilization of bandwidth [5], traffic grooming has to be used to support statistical multiplexing for better bandwidth utilization [6].

1.2.2 OPTICAL PACKET SWITCHING (OPS)

In OPS architecture, user data along with the control information is transmitted in optical domain. At each core node control information is extracted and processed in electrical domain, user data packets remains in optical domain and routed on the basis of information processed at the core nodes, no optical-electrical (O-E) and electrical-optical (E-O) conversion is required for the user data. However control information passes through (O-E-O) conversion.

As a lack of needless (O-E-O) conversion at the core nodes, OPS provide full transparency, high efficiently and increased scalability. Because of these reasons, OPS is considered as a promising candidate for high bit rate data across optical networks of the future [7].

However, in the current optical communication technology the main problem with OPS is the lack of optical buffers to resolve output contention. There is an additional problem with OPS is the delay experienced by the incoming data packets for the configuration of switch [8].

1.2.3 OPTICAL BURST SWITCHING (OBS)

In OBS, the basic data transmission unit is data burst. Data burst consist of several data units of IP packets, ATM cells or row bits stream for a specific destination. A control packet is sent with an offset delay before transmitting the data burst to reserve the network resources from source to destination (establishing a light path for short duration).

OBS network consist of best of features of OCS and OPS networks that’s why it has the potential to be the most practical all optical network, AON [9]. Silent features of OBS are listed below

- Data gets aggregated at the ingress node to form a data burst.
- The light path established between the source to the destination is short lived till the burst is transmitted, unlike OCS.
- Basically OBS should be a buffer less optical network but to reduce burst loss FDL and wavelength converters can optionally be used.

- Control information and data bursts are carried on separate wavelengths.

2 Insight to OBS network

OBS is an optical network switching technology which is designed to attain equilibrium between optical circuit switching and optical packet switching networks. In an OBS network data burst consisting of multiple data packets is switched through the network all optically. A control header is transmitted ahead of data burst in order to configure switches along the burst's route.

Network Architecture

An optical burst switched network consists of optical switching nodes which are connected through optical fibers. In an OBS network, nodes can either be edge nodes or core nodes as shown in figure 2.1. Edge node is responsible for aggregating packets from the access network into a data burst and scheduling of data burst for transmission on outgoing wavelength channel. The primary responsibility of a core node is the switching of data burst from the incoming port to the outgoing port and wavelength contention handling.

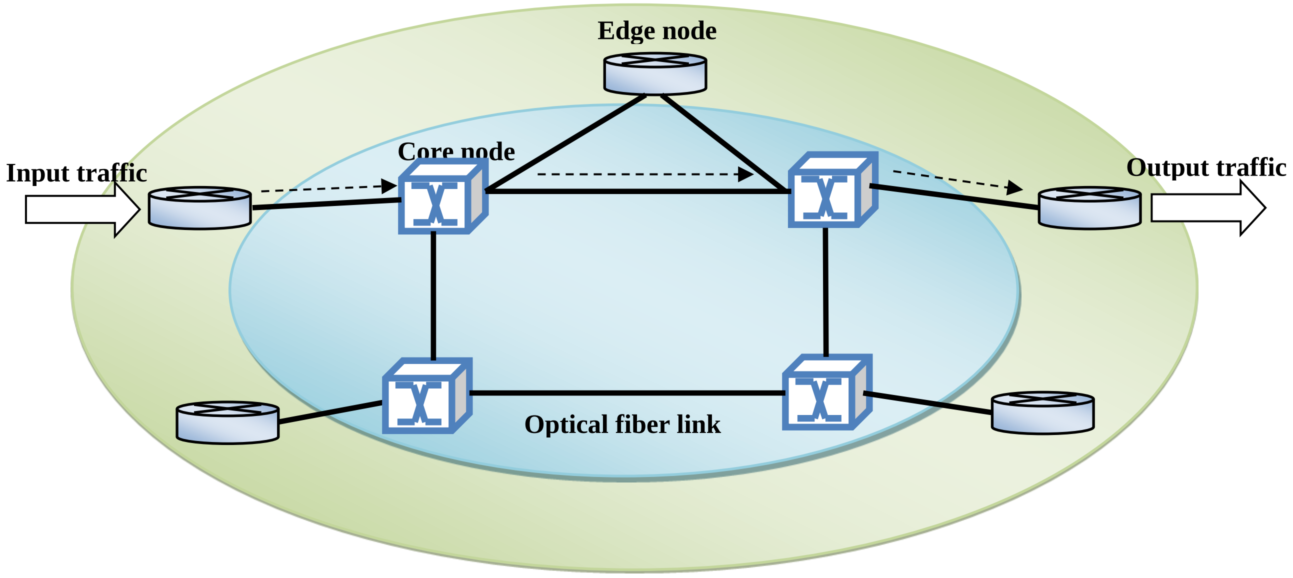


Figure 2.1: Optical Burst Switched Network architecture

Data packets from the access network (which is the client data of various types) are aggregated at the ingress node (edge node) and data for the same destination get aggregated to form a data burst. The aggregated data burst is then transmitted to the destination as shown in the figure 2.2 (a). Data burst will be disassembled later at the egress node as shown in figure 2.2 (b). The client data is buffered at the edge node in the electronic RAM [10].

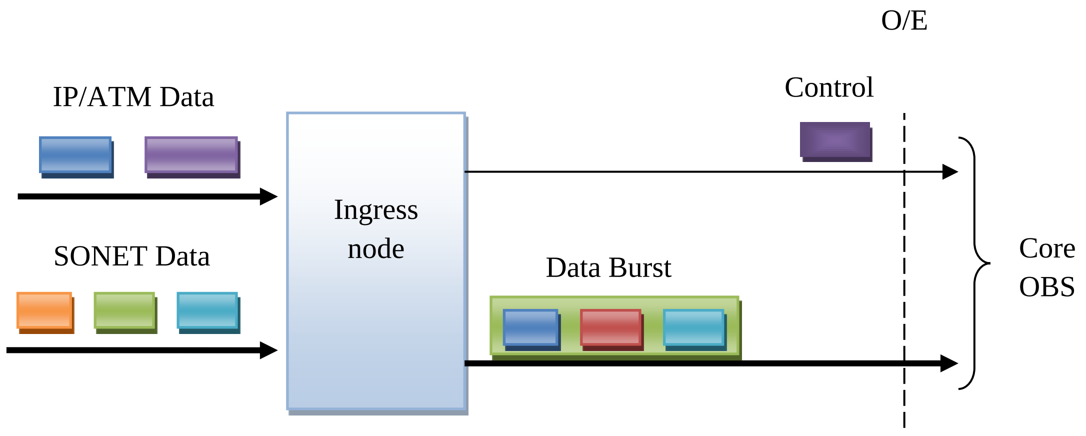


Figure 2.2 (a): Burst Assembly process of OBS

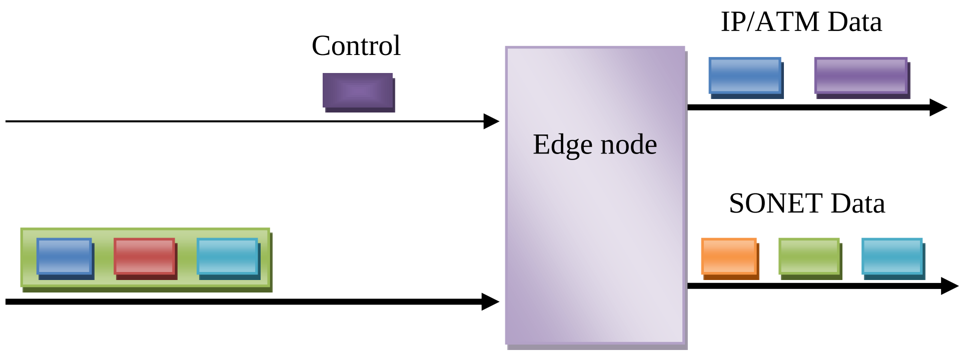


Figure 2.2 (b): Burst Disassembly process of OBS

In the figure 2.2 the data and control planes are separate within the core OBS network. There is a dedicated channel to transmit the control information, which carries usual header information, for each data burst. The control packet is very small in size so a single dedicated channel is sufficient for multiple data channels. Control packet goes through O/E/O conversion at each OBS intermediate nodes. The conversion is necessary for the processing of the information which this control packet has and on the basis of this processed information, configuration of the intermediate node is performed for data bursts. Offset time is injected between the control packet and data burst transmission to compensate the processing and configuration delay. Offset time should be chosen appropriately that control packet configure all the intermediate nodes before the arrival of the data burst. In such situation no O/E/O conversion is required (RAM) for data burst and the use of FDLs to save the data burst dropping [10, 11].

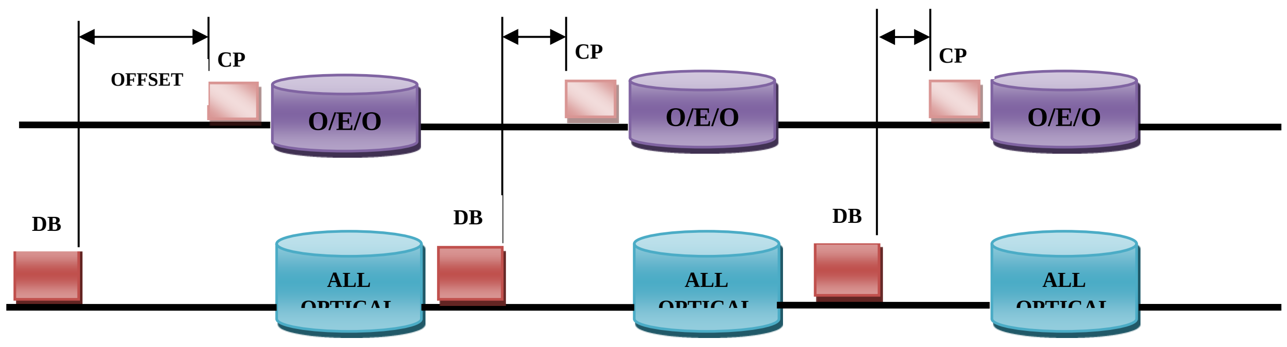


Figure 2.3: Data and Control packet in an OBS network

2.1 At the edge of the OBS network

2.1.1 Burst aggregation

Bust aggregation is the process of assembling data packets from the upper layers, sorting according to the destination addresses and then aggregating to form a data burst. Burst aggregation may affect the overall operation of the network, for example, if large size data burst is aggregated for that network resources will also be held for a longer period of time. The vital parameter in data burst aggregation is the trigger criteria for determining when to assemble a data burst. There are three main burst aggregation proposed techniques which are time based approach [11], length based [11] and hybrid of two mentioned approaches [12].

2.1.1.1 Time based approach

In this approach a preset timer is set and data of specific destination is aggregated up to the expiry of the timer. As timer expires the data burst is ready to be transmitted to the desired destination. To ensure the minimum length of the data burst, minimum length limitation is applied [13]. Padding is applied if the burst length is lower than the defined burst length. In case of heavy network traffic, this approach aggregates larger data bursts, which is one of the limitations of this proposed scheme.

2.1.1.2 Length Based approach

In this approach data is aggregated up to a defined maximum length of the burst. Data from higher layers is collected to form of burst in terms data size limit instead of time limit. Network resources will be held for a longer period of time in a scenario where the traffic arrival rate is low because burst aggregating and sending will be slow which affects the overall performance.

2.1.1.3 Hybrid approach

In this hybrid proposed technique the limitations of the above mentioned approaches have been removed [12]. A timer is set and when the data aggregation is reached to the defined time, the aggregated data is then compared with minimum and maximum defined length of the burst, if the aggregated data is larger than the minimum defined length the data is packed into a single burst and transmitted else timer is restarted till the size of burst is larger than the minimum defined length (size). In case the aggregated data is larger than the maximum defined length than one data burst is formed of the maximum length and the remaining data is added to the next coming burst.

2.2 Signaling protocol for OBS

Signaling protocols must be implemented in an OBS network to allocate the network resources and configure the switches for a data burst at each node. Out of band scheme is adopted to implement signaling scheme in an OBS network. In out of band scheme, a control packet which is associated with a specific data burst is transmitted over a different wavelength. This control packets travel along the same route as the data burst, informing each node to configure the cross connects and accommodates the arriving data burst at appropriate time.

In other words the signaling protocols in OBS networks are proposed to specify connection requests for the communication of the nodes. Signaling protocols also ensures that whether or not the resources of the network are efficiently utilized. Several signaling techniques have been proposed in the OBS literature, few popular techniques are discussed here.

2.2.1 Centralized signaling with end to end reservation

This approach is proposed in [14], the authors have proposed a centralized end to end resource reservation mechanism. In this approach a centralized request server is established which is responsible for the establishments of the data burst transmission routes and allocating wavelengths to these routes for every source to destination pair in the network. Thus network resources are reserved from source to destination by the central request server inside the OBS network. The global knowledge of all the switches states inside the OBS network is carried by this central server. This proposed technique may perform efficiently for smaller networks.

2.2.2 Distributed signaling with one way reservation

In this proposed approach the control packet is sent before the transmission of the data burst, the control packet carries all the information regarding the data burst such as data burst size, data arrival time etc. The purpose of transmitting control packet before data burst is to configure the nodes and reserve the network resources, figure 2.4 illustrates this scenario where control packet is configuring intermediate nodes and reserving network recourses. Information about the data burst in the control packet is processed electronically at the intermediate nodes. Data burst is sent after an offset delay without waiting for positive acknowledgement of the reserved resources. The limitation of this proposed technique is that there is a chance that control packet may not reserve the resources successfully in such scenario the data burst will be dropped.

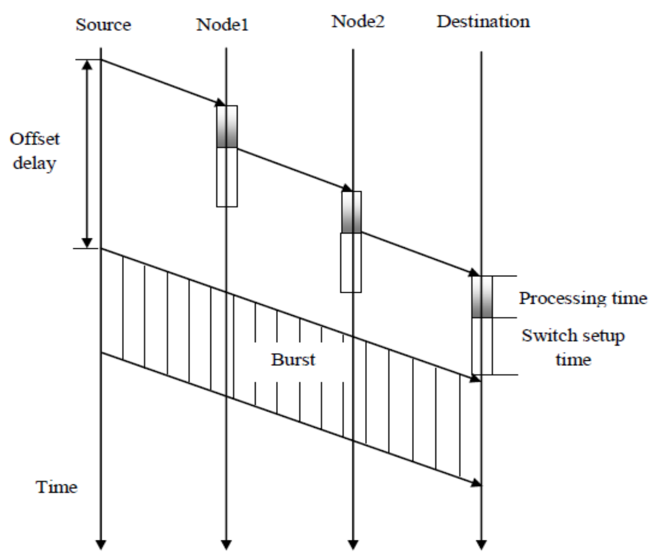


Figure 2.4: Distributed Signaling one way reservation

2.2.3 Distributed signaling with two way reservation

This proposed approach is acknowledge based and works on the same principle that the control packet carrying the data burst information is sent into the network before the burst transmission. The data burst waits until positive acknowledgement is received to the sender from the receiver. The positive acknowledgement confirms that network resources are reserved. If any of the intermediate node is unable to allocate resources to a data burst, negative acknowledgement is sent to the source by the intermediate node and data burst is not transmitted, as network resources are not available. This approach does not cause any data burst dropping but it badly affect the overall performance of the network as the data burst needs to wait for a longer period for the acknowledgment.

2.3 Resource reservation schemes in OBS

1. TELL-AND-GO (TAG) scheme [15]

2. RESERVE a FIXED DURATION (RFD) scheme [16].

In TAG, the channel (wavelength) is assigned as the data burst ready for the transmission. In RFD, control packet reserve the channel (wavelength) for specific time interval, the duration should be ample enough that the burst reach to the destination.

JUST IN TIME (JIT)

The JIT protocol is based on TAG. When a control packet is arrived at the OBS node the wavelength is immediately reserved for the data burst and incase no wavelength is available then the request is blocked and that data burst is dropped. In case of successful reservation of the resources the light path remains establishes till data transmission is finished. In JIT, time is divided into periods during which the wavelengths are reserved and allocated to the transmitting burst. In [20] the author has divided JIT schemes based on the amount of time a burst occupies a path, these basic schemes are discussed below.

Explicit setup & explicit release: Wavelength is reserved and cross connects are configured immediately when the control packet is processed and the source sends a control packet for the release of the network resources. This means that the burst transmission is ended.

Estimated setup & explicit release: The reservation of wavelength & configuration of cross connect is delayed until the actual burst is arrived and the source sends a control packet for the release of the network resources. This means that the burst transmission is ended.

Explicit setup & estimated release: Wavelength is reserved and cross connects are configured immediately when the control packet is processed and the node can calculate the end of burst transmission by considering burst length, thus node knows when to release the resources.

Estimated setup & estimated release: The reservation of wavelength & configuration of cross connect is delayed until the actual burst is arrived and the node can calculate the end of burst transmission by considering burst length, thus node knows when to release the resources.

In [19] the author has described a scheduling algorithm, called Horizon. In horizon, explicit - estimated release is used as the control packet have the information about the burst length and the offset time; from which the node can estimate that when to free the resources. It is explicit in nature as upon the arrival of the control packet the resources are reserved and maintain a horizon. Following are some of the algorithms that are proposed.

JUST ENOUGH TIME (JET)

JET follows RFD scheme (one way reservation scheme), where control packet carries the data burst length information and reserve the network resources for fixed duration which is exactly equals to the burst transmission time. In JET control packet is transmitted ahead of the corresponding data burst that the control packet to be processed at each intermediate nodes and establishes an all-optical data path for the corresponding data burst. On the successful reservation, switch will be configured prior to the burst arrival. Data burst is transmitted optically after a predetermined offset delay. The offset delay is determined by knowing the intermediate hops between source and destination and switching time at each core node. The control packet carries additional information like destination address, the wavelength associated with data burst and the burst offset time along data burst length.

2.4 Scheduling Algorithms in OBS

When a control burst arrives at a node, a wavelength channel-scheduling algorithm is used to determine the wavelength channel on an outgoing link for the corresponding data burst. The information required by the scheduler such as the burst's arrival time and its duration are obtained from the control burst. The scheduler keeps track of the availability of the time slots on every wavelength channel. Scheduling of data is different in OBS network than IP based network, as in IP based network data is stored in electronic buffer and scheduled to the desired output port. In optical burst switched network, when data arrives at the core network it must be sent to the next node without storing into electronic buffer, as data have to remain in optical domain.

In an OBS network, if necessary FDL can be used at the core nodes. The scheduler selects one or more FDLs to delay the data burst. A wavelength channel is said to be unscheduled at time ***t*** when no burst is using the channel at or after time ***t.*** A channel remains unused in the duration of voids, which occurs between successive bursts and after the last burst assigned to the channel.

Several issues affect the performance of the OBS scheduler. First, it must select wavelength channels and FDLs in an efficient way to reduce burst dropping probability. Secondly, it must be simple enough to be able to handle large number of bursts in a very high-speed environment. Furthermore, the scheduler must not lead to early data burst arrival situation, in such scenario; the data burst will arrive before the control burst has been processed.

In this section we have discussed several scheduling algorithms which proposed in OBS literature.

2.5.1 FIRST FIT UNSCHEDULED CHANNEL (FFUC) Algorithm [21]

For each of the outgoing wavelength channels, the FFUC algorithm keeps track of the unscheduled time. Whenever a control burst arrives, the FFUC algorithm searches all wavelength channels in a fixed order and assigns the burst to the first channel that has unscheduled time less than the data burst arrival time. This algorithm's main advantage is its computational simplicity. Its main drawback is that it results in high dropping probability, as the algorithm does not consider voids between bursts scheduling.

2.5.2 LATEST AVAILBLE UNSCHEDULED CHANNEL (LAUC) Algorithm [21]

The basic idea of LAUC algorithm is to increase channels utilization by minimizing voids created between bursts. This is accomplished by selecting the latest available unscheduled data channel for each arriving data burst. For example, consider wavelength 1 and 2 are unscheduled at time ***t'***, and wavelength 1 will be selected to carry the new data burst arriving at ***t'***. In such scenario, the void on wavelength 1 will be smaller than the void that would have been created if wavelength 2 were selected. Therefore, LAUC yields better burst dropping performance than FFUC algorithm while it does not add any computation overhead. However, since it does not take advantage of voids between bursts, as was the case for the FFUC, it still leads to relatively high dropping probability.

2.5.3 LAUC with VOID FILLING (LAUC-VF) Algorithm [22]

The LAUC-VF algorithm is a variation of LAUC algorithm except that voids can be filled by new arriving bursts. The basic idea of this algorithm is to fill the voids by selecting the latest available unused data channel for each arriving data burst. Given the arrival time ***t'*** of a data burst with duration L to the optical switch, the scheduler first finds the outgoing data channels that are available for the time period of (***t'; t' + L***). If there is at least one such data channel, the scheduler selects the latest available data channel, i.e., the channel having the smallest gap between ***ta*** and the end of the last data burst just before ***ta***. Since the voids are used effectively, the LAUC-VF algorithm yields better performance in terms of burst dropping probability than FFUC and LAUC algorithms. On the other hand, the algorithm is more complex than FFUC and LAUC algorithms because it keeps track of two variables instead of one.

2.5.4 GENERALIZED LAUC-VF (G-LAUC-VF) Algorithm [22]

G-LAUC-VF algorithm generalizes the LAUC-VF algorithm to include QoS features. At core switches, the scheduler associated with a link will schedule data bursts (DB) going to that output link. For each link, its scheduler maintains and queues Q_1, Q_2..Q_n, with Q_i being used to store the control packet of class ***i*** in first in first out (FIFO) order.

2.6 TIME SLOTTED OPTICAL BURST SWITCHING (TS-OBS)

TSOBS is a variant of OBS network, proposed first time in [23]. TSOBS and OBS networks follow the same control information signaling scheme by transmitting the control packet before the data burst transmission. In TS-OBS, all data bursts are of same length and these fixed length DBs are transmitted in fixed time slot.

Optical networks lack optical buffers and wavelengths cannot be scheduled for all transmitting data burst thus optical networks lack fair scheduling. As in traditional fair channel scheduling algorithms, if packet comes and the channel is not available it is then buffered for the fair scheduling. To achieve fair scheduling of channels in TSOBS, wavelengths have been allocated to the time slots. These time slots ideally equal to the propagation delay a burst experience in reaching to the destination or the propagation delay of link should be an integer multiple of the time slot [24,25] for synchronization purpose.

2.6.1 Network Architecture of TSOBS

This network architecture is proposed in [25], where allocation of resources is done through centralized network control architecture. ITU-T follows centralized control architecture, where data and control plan are separate [26], to ensure synchronization in TSOBS separate planes are established.

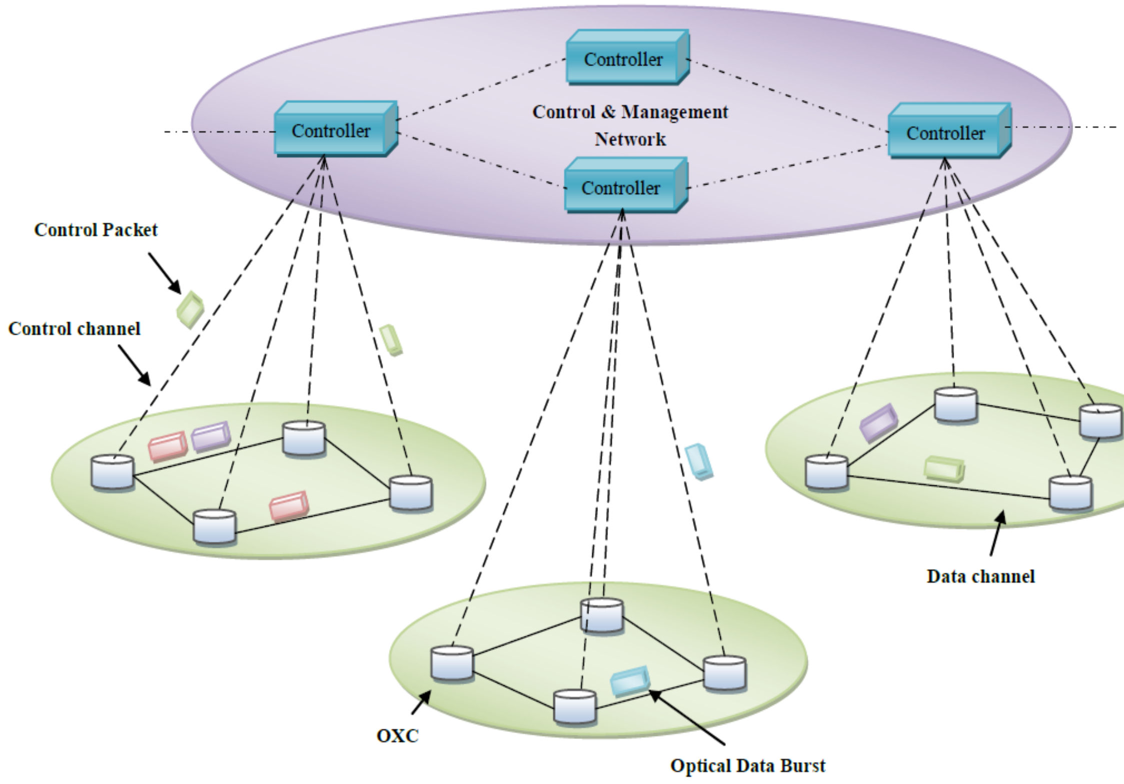


Figure 2.5: Centralized network control architecture

In this architecture, OXCs are connected to a controller. Here to reduce the processing load on the controller, limited numbers of OXCs are connected to a single controller. These controllers are connected to each other through optical medium to form network control and management network. These controllers have topology and routing information.

In [25] when there is burst to be transmitted, OXC sends a control packet to the controller for the allotment of time slot for the transmission of DB. This process incurs additional delay, resulting in the overflow of edge nodes buffers (data packets dropping) and delay DB transmission.

2.7 Wavelength contention in OBS

Wavelength contention occurs in OBS network when same wavelength is assigned by the scheduler to more than one data bursts, which results in the dropping of data bursts. Drop rate of data bursts is one of the important performance parameter of the OBS network. In traditional IP based networks contention can be controlled through electronic buffers; however, in optical network there is no equivalent optical random access memory.

2.7.1 Optical buffering

Optical random access memory is not available yet but data burst can be delayed for fixed amount of time by utilizing fiber delay lines (FDLs). By utilizing FDLs in parallel or in stages, data burst can be delayed for variable amount of time. Thus after a specific amount of delay the outgoing port will be free to for data burst transmission to the next node. The proposed contention avoidance techniques based on FDLs results in lower data burst droppings but does not guarantee the correct ordering of the data packets. The size of optical buffer is limited not only by signal quality concern but also by physical space limitation. To delay a single data burst for 1 ms delay line of over 200 km is required (considering 2×10^8 m/s signal speed).

2.7.2 Wavelength conversion

Wavelength conversion is the process of converting the wavelength of an incoming channel to different wavelength to an outgoing channel. The device which performs this operation is called wavelength converter. Let us suppose, two data burst are destined to the same output channel the contention can be avoided if both the data bursts transmitted on different output wavelengths. Thus wavelength conversion technology has the capability to avoid contention. This technology is expensive and not yet mature enough, demonstrated in laboratory environment. The range of wavelength conversions is somewhat limited and the categories of wavelength conversion are listed below.

Full conversion: Any incoming wavelength can be converted to any outgoing wavelength.

Limited conversion: In this wavelength conversion technique conversion is limited and not all incoming wavelengths can be converted to any outgoing wavelength channels. Here the cost of the switch is reduced by applying the restriction.

Fixed conversion: This technique again offers limited conversion where incoming wavelength is converted to two or more predetermined outgoing channel.

2.7.3 Deflection routing

One of the proposed techniques to resolve wavelength contention is by routing the contented burst to a different outgoing port other than the intended port. This technique is called deflection routing. Deflection routing is not preferred in electronic packet switched networks because of out of sequence delivery of data packets but can be preferred in optical networks where buffering is limited. In deflection routing, the data burst usually takes a longer route which results in increased delay and degradation of signal quality.

2.7.4 Burst segmentation

When data bursts contend they do not completely overlap each other, usually portion of data bursts overlap. Instead of dropping the whole data burst it is desirable to drop the contented portion. This technique is called burst segmentation. Burst segmentation minimizes data packets dropping by portioning the data burst in to segments and drop the contented segments. Segments are the basic transport unit of a data burst which carries segment header and payload. The segment payload may carry any type of data such as ATM frames, IP packets etc. The size of the segment is a key system parameter; the segment size can be fixed or variable. Fixed size segments helps in better synchronization at the receiving end while variable length segment can accommodate variable length data in efficient manner.

Segment size also offer tradeoff between loss per contention and segment overhead per burst. Longer segment will results in high loss per contention but low segment overhead.

Many techniques have been proposed to minimize wavelength contention by adopting the above discussed techniques as a foundation stone. Some of the recent research work in wavelength contention is discussed below.

In [27], the authors has proposed an algorithm Absolute Fair Quality of Service Differentiation Scheme, AFQD, which guarantees loss-free transmission for high priority data bursts, for any kind of topology of the OBS network. AFQD is based on a wavelength partitioning scheme, called optimization topology aware wavelength partitioning scheme, OTWP, which uses integer linear programming to partition data wavelengths among the nodes in the network.

In [29] an attempt is made for the complete isolation of the classes by forming the contours of the bursts of all classes and on the basis contours formed for each class, priority is given to the class with delay sensitive data. Thus data burst dropping rate for the delay sensitive class is minimized as channel is preempted from the lower class for the higher class and complete isolation is also achieved using this technique.

In order to ensure QoS and minimize burst dropping in an OBS network, a FDL bank is used and a dynamic FDL bank partitioning algorithm is proposed in [30], they have analyzed the feasibility conditions for the dynamic FDL bank partitioning algorithm.

3 Overview of the work

We propose a time sliced OBS network architecture. The aim of this solution is to minimize wavelength contention and improve packet level performance at the edge node. Separate physical and synchronization planes are defined for the synchronization of all core nodes. Modulo arithmetic is used to minimize wavelength contention. Different time slots are associated with wavelengths for the establishment of light path using modulo arithmetic.

3.1 Network architecture

Our proposed network architecture is based on overlay network model, that is, we have employed a synchronization network over a physical network. The synchronization network is responsible for the synchronization of all core nodes of the OBS network. All core nodes are divided into groups and clocks in synchronization plane are synchronizing these groups by sending sync packets after specific intervals. These clocks are synchronized so that all the core OBS nodes send and receive data bursts in synchronized manner.

Every clock unit has an associated wavelength. Once the data burst has been aggregated at the edge node on any clock unit, a wavelength will be selected which is associated with that unit of clock. This selected wavelength will be used for the establishment of light path from the source to the destination. The life of a light path depends upon the transmission time of a data burst to reach to the destination after that light path will be expired. Here comes the importance of defining time slots, considering the transmission time of a data burst to reach to the destination, slot will be allotted for the data burst delivery. With the expiry of time slot light path also expires and the wavelength which was used for the establishment of the light path can be used for some other light path establishment.

There might be a possibility that the same wavelength is used for the establishment of different light paths at the same time which results in wavelength contention. Wavelength contention will cause data burst dropping. We have used modulo table, a branch of number theory to avoid wavelength contention in the network.

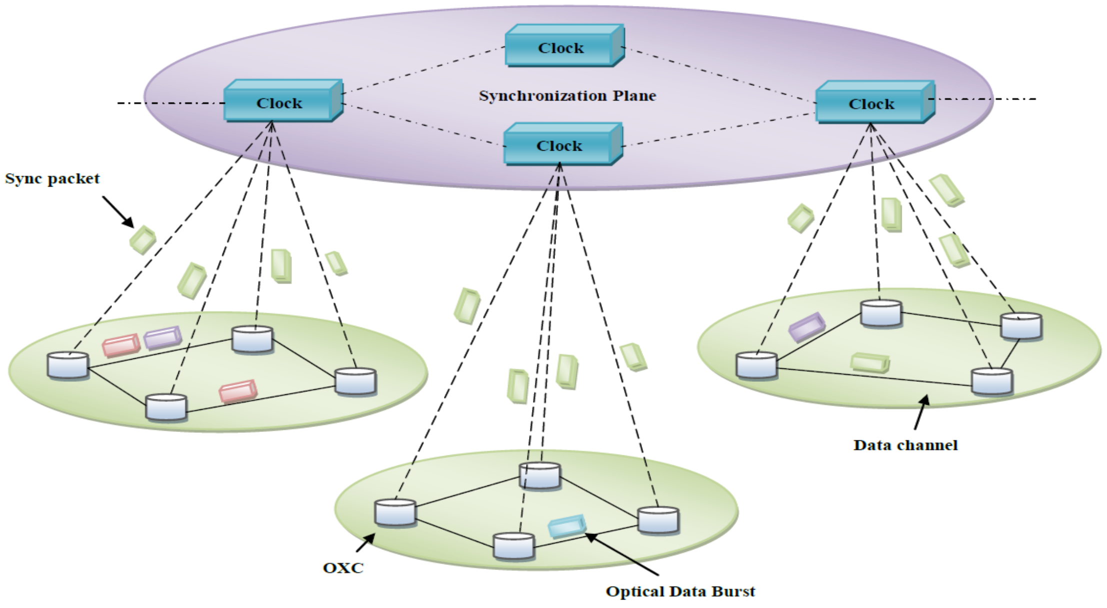


Figure 3.1: Proposed Network Architecture

3.2 Association of time slots with wavelengths using modulo arithmetic (Network Clock synchronization)

Core nodes are synchronized with the help of synchronization (control/management) plane. All nodes have equal number of wavelengths for the establishment of light path. A sample modulo table is associated with wavelengths and clock slots, as shown in table.1. When a data burst is aggregated at the edge node, the core node first check the clock, for example, a data burst is ready for the transmission at the core node 1, if the clock unit is 2 the wavelength λ_2 will be assigned for the establishment of light path from source to destination. Similarly if node 3 has a data burst to transmit at clock unit 1 then λ_3 will be used for the light path establishment. Importance of using modulo arithmetic is that at any clock unit there is no wavelength contention as at any clock unit all nodes will use different wavelengths. In table.1 it is assumed that specific data burst requires only one time slot to reach to the destination.

Clock slots	**Number of wavelengths with node 1**	**Number of wavelengths with node 2**	**Number of wavelengths with node 3**
1	λ_1	λ_2	λ_3
2	λ_2	λ_3	λ_1
3	λ_3	λ_1	λ_2

Table.1: Sample slot allocation table

In the figure 3.2 a sample network topology is shown to understand the working concept. In this network topology edge nodes are connected with the core OBS nodes. When data packets get aggregated into a data burst for any specific destination, clock is checked by the source node and data burst is transmitted. It is assumed that two adjacent nodes are at unit length from each other.

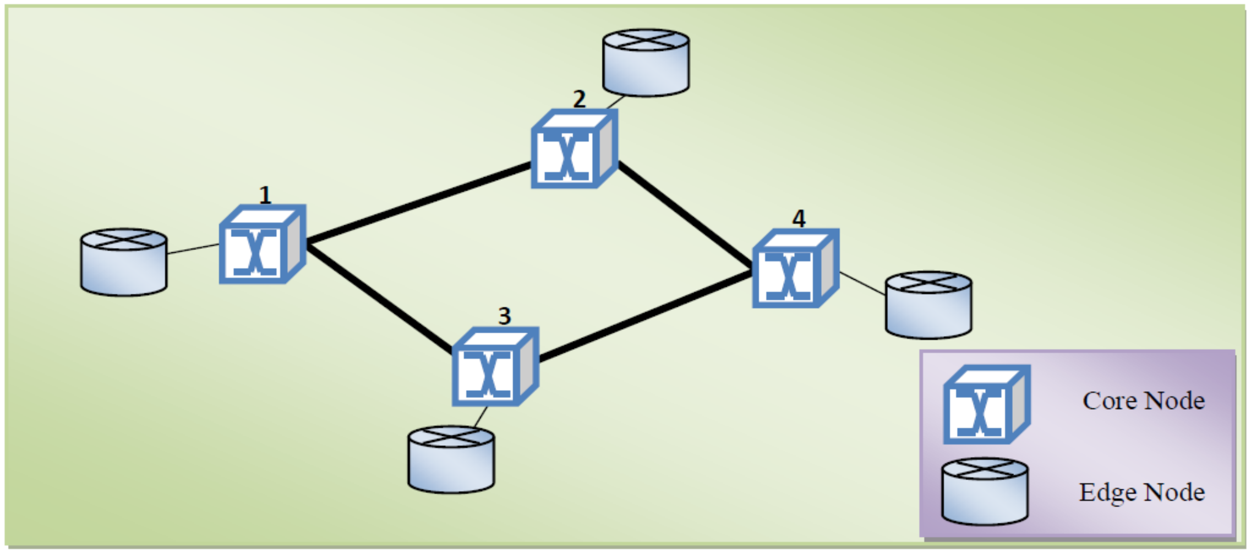


Figure 3.2: Sample network topology

Figure 3.3 is presenting the time slot requirement of the sample network topology in figure 3.2, that is, how many units of clock will be required for a data burst to reach to the destination. On the basis of the shortest path routing total units of clock required for all links in the topology is calculated.

Node 1	
Source to Destination	Clock units required
1-2	1
1-3	1
1-4	2

Node 2	
Source to Destination	Clock units required
2-1	1
2-3	2
2-4	1

Node 3	
Source to Destination	Clock units required
3-1	1
3-2	2
3-4	1

Node 4	
Source to Destination	Clock units required
4-1	2
4-2	1
4-3	1

Figure 3.3: Clock units requirement of above mentioned network topology

Partial routing information is shown in the figure.3.3 above and on the basis of this information slot allocation table.2 (based on modulo table) is shown below.

Clock slots	Number of wavelengths with node 1		Number of wavelengths with node 2		Number of wavelengths with node 3		Number of wavelengths with node 4	
	For longer transmission delay slots	For smaller transmission delay slots	For longer transmission delay slots	For smaller transmission delay slots	For longer transmission delay slots	For smaller transmission delay slots	For longer transmission delay slots	For longer transmission delay slots
1	λ_1	λ_5	λ_2	λ_6	λ_3	λ_7	λ_4	λ_8
2		λ_6		λ_7		λ_8		λ_5
3	λ_2	λ_7	λ_3	λ_8	λ_4	λ_5	λ_1	λ_6
4		λ_8		λ_5		λ_6		λ_7
5	λ_3	λ_5	λ_4	λ_6	λ_1	λ_7	λ_2	λ_8
6		λ_6		λ_7		λ_8		λ_5
7	λ_4	λ_7	λ_1	λ_8	λ_2	λ_5	λ_3	λ_6
8		λ_8		λ_5		λ_6		λ_7

Table 2: time slot and corresponding wavelength allotment table

This is the wavelength allocation table for the above mentioned sample network topology in figure 3.2, table.2. In this table wavelengths for longer transmission delay are separated from the wavelengths for smaller transmission delay. Every node has equal number of wavelengths i.e 8. Using this table we can ensure no wavelength contention as if any node wants to send data burst to any destination no wavelength contention will occur. If node 1 wants to transmit data burst to node 4 at clock unit 1, it requires 2 slots to reach to the destination; λ_1 will be assigned for this burst. λ_1 will be reserved for two clock units which is considered as one time slot for longer transmission delay data burst transmission. If any node wants to send data burst to a destination which requires one clock unit, for example, node 4 wants to transmit data burst to node 2 at clock unit 7; in such scenario λ_6 will be assigned for the establishment of light path. This pattern is repeated as it is the inherit property of modulo table, after the elapsing of eighth clock unit, the whole pattern will be repeated and will start from clock unit one.

Actual network topology links have different transmission delays. It is unrealistic to assume that all links have the same transmission delays. To make our solution more realistic we are assuming national science foundation, NSF, Network topology.

3.3 Implementation of proposed solution on 14 nodes NSFNet topology

Routing inside the core network is based on Dijkstra's shortest path routing algorithm which uses physical distance as a metric. Weights are assigned to each link according to the actual distance between the links in km. A 14 nodes NSFnet network topology is shown in the figure 3.4 with the link weights.

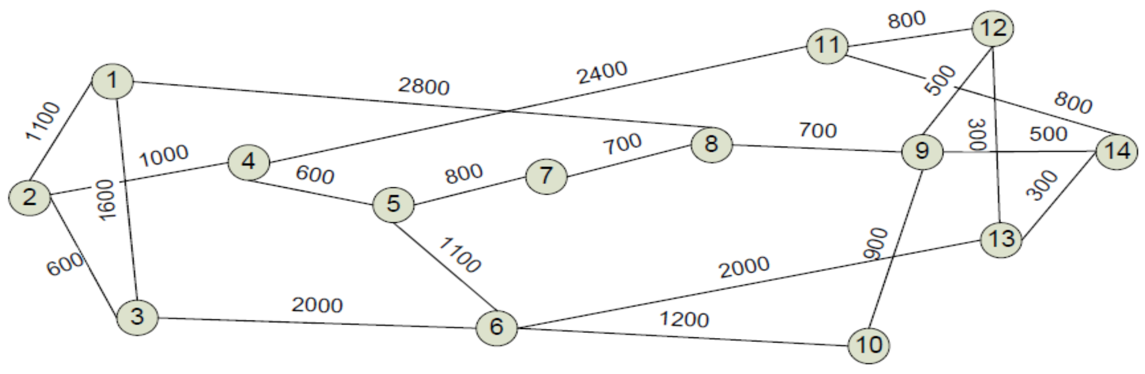


Figure 3.4: 14 nodes NSFNet network topology

By considering the shortest path routing, shortest distances have been calculated among all nodes. If a data burst is transmitted from any node to any destination in this network topology transmission time is calculated on the basis of 1 [31]. The switching time of core nodes is taken 0.01 ms [32] and speed of light inside the optical fiber is 2×10^8 m/s [31].

$$\boldsymbol{Time_{transmission} = Time_{assembly} + (N_{hops} \times Propagation\ time) + Time_{disassembly}} \quad (1)$$

The smallest transmission time is 1.51 ms and highest transmission time is 22.51 ms according to the equation. The derived time slot and corresponding wavelength allotment table is shown in table 3. We have calculated the transmission delay of all links for the reason that we classify transmission delay into multiple classes and every class has separate time slot. The clock unit of classes is the arithmetic progression of the selected slot size (for example, if the selected slot size is 1.51, the clock unit of first class will be 1.51 ms , second class will be 3.02 ms and so depending upon the number of classes) . The clock unit (slot size) should be a complete divisor to the highest transmission time of the topology to give zero remainder. The number of classes of wavelength can be calculated by using equation number (2).

$$\boldsymbol{Number\ of\ classes\ of\ wavelengths = \frac{Highest\ Transmission\ time\ of\ the\ topology}{Slot\ size\ (Complete\ divisor)}} \quad (2)$$

Clock slots	Number of wavelengths available with node x		
	For smaller transmission delay slots	For medium transmission delay slots	For larger transmission delay slots
1	λ_1	λ_{15}	λ_{29}
2	λ_2		
3	λ_3	λ_{16}	
4	λ_4		λ_{30}
5	λ_5	λ_{17}	
6	λ_6		
7	λ_7	λ_{18}	λ_{31}
8	λ_8		
9	λ_9	λ_{19}	
10	λ_{10}		λ_{32}
11	λ_{11}	λ_{20}	
12	λ_{12}		
13	λ_{14}	λ_{21}	λ_{33}
14	λ_1		
15	λ_2	λ_{22}	
16	λ_3		λ_{34}
17	λ_4	λ_{23}	
18	λ_5		
19	λ_6	λ_{24}	λ_{35}
20	λ_7		
21	λ_8	λ_{25}	
22	λ_9		λ_{36}
23	λ_{10}	λ_{26}	
24	λ_{11}		
25	λ_{12}	λ_{27}	λ_{37}
26	λ_{14}		
27	λ_1	λ_{28}	
28	λ_2		λ_{38}
29	λ_3	λ_{15}	
30	λ_4		
31	λ_5	λ_{16}	λ_{39}
32	λ_6		
33	λ_7	λ_{17}	
34	λ_8		λ_{40}
35	λ_9	λ_{18}	
36	λ_{10}		
37	λ_{11}	λ_{19}	λ_{41}
38	λ_{12}		
39	λ_{14}	λ_{20}	
40	λ_1		λ_{42}
41	λ_2	λ_{21}	
42	λ_3		

Table.3: Allocations of wavelengths on the basis of transmission time

In table.3, transmission time is divided into three categories smaller, medium and longer. The value of one clock unit is equal to 7.5 ms for this table and total numbers of wavelengths used are 42 to provide contention free environment. If we take smaller values of clock unit more number of wavelengths will be required, for example, if one clock unit time is taken equal to 4.5 ms then to accommodate the highest transmission time we will need to divide the transmission time into five categories and as each category must have 14 wavelengths so total of 70 wavelengths will be used. Mathematical formulation is given in (3).

$$\boldsymbol{Total\ number\ of\ wavelengths = Number\ of\ classes\ of\ wavelengths\ x\ Number\ of\ nodes\ in\ a\ network} \quad \textbf{(3)}$$

3.4 Burst aggregation

At the edge nodes of an OBS network, burst aggregation is performed. The burst aggregation technique which is used in this proposed solution is timer based data burst aggregation technique [33]. The reason of selecting timer based aggregation technique is to synchronize the data packets aggregating with data burst transmission within the network. In figure 3.5 data aggregation is shown at the edge node, data of different colors are depicting to data packets for a specific destination and each destination has a dedicated buffer. Destination buffers are distinguished by different colors. Each buffer is basically divided into two sections one for holding the data packets, which is referred as holding buffer in our proposed solution and the second one is sending buffer where burst is aggregated. Time slots are allocated to the data burst on basis of transmission delay from source to the destination. As each destination has separate buffer so it gets easy to program the allotment of time slots according to transmission delay to reach the data burst to the destination.

We have introduced guard time to compensate the delay while DB is aggregated and ready for the transmission as depicted in (4).

$$\boldsymbol{Data\ burst\ Aggregation\ time = Slot\ size - Guard\ time}(\Delta) \qquad (4)$$

The data packets coming from the access network get aggregated into a data burst for transmission as shown in figure 3.5. The formation of data burst depends on the transmission delay a data burst requires to reach to the destination from the source. This figure 3.5 depicts the scenario of burst aggregation and the size of the burst is equal to the size of the slots.

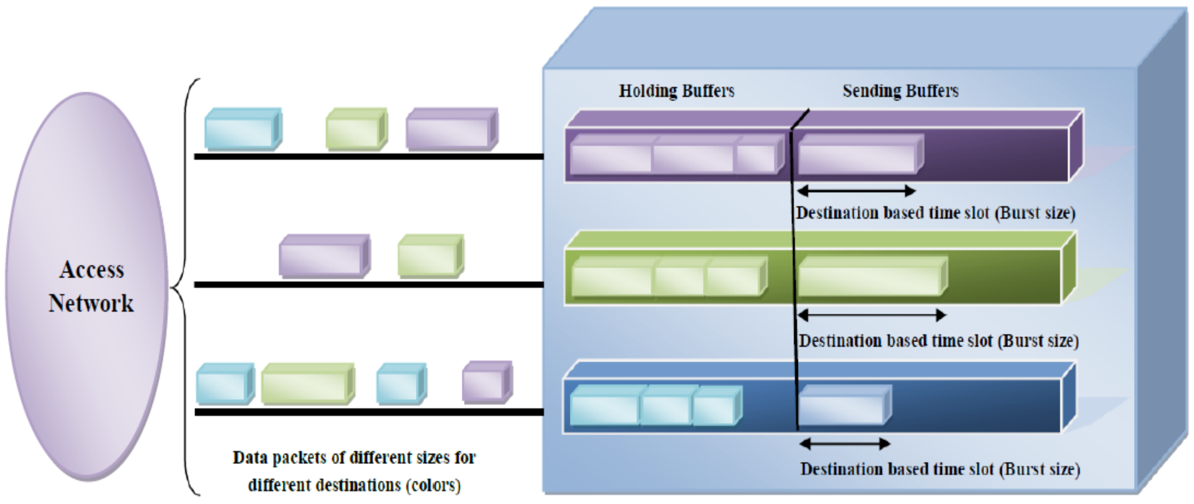

Access Network
Data packets of different sizes for different destinations (colors)
Holding Buffers
Sending Buffers
Destination based time slot (Burst size)
Destination based time slot (Burst size)
Destination based time slot (Burst size)

Figure 3.5: Edge node buffer architecture and data burst aggregation based on transmission delay

3.5 Modulo table generation for N nodes network topology

Pseudo code is to generate modulo table for N nodes network. This code is a generalized version for any network topology.

```
BEGIN
    FOR i = 0 to N(total number of nodes in a network)
    {
      FOR j = 0 to N
            k = i + j;
            $\lambda_j$ = k MOD N
            IF k >= N THEN $\lambda_j$ = k-N MOD N
}
END
```

3.5 Proposed Algorithm

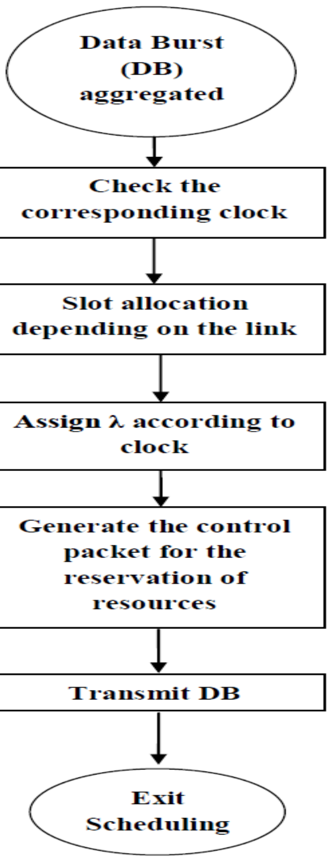
Data packets get aggregated for a specific destination to form into a single data burst. The burst aggregation is timer based in our proposed solution and the value of timer is equal to the slot size. In our solution the slot size depends on the transmission delay for a specific destination. When data burst gets aggregated for a specific destination, the slot size required to that data burst is allotted which depends upon the transmission delay from source to destination. We have classified the transmission delay into two categories (depending on the topology used) termed lower link delay and higher link delay. Depending upon the transmission delay fixed number of required slots from a specific category is allocated to the data burst and wavelength associated to that slot will be reserved for these slots until that slot time expires. When slot and wavelength are assigned to the data burst a control packet is generated to carry information (burst length, wavelength associated etc) to inform all intermediate nodes and destination node. With offset delay value data burst is transmitted to the destination.

3.5.1 Pseudo Code

```
BEGIN
Check the link status (Lower or Higher delay link)
        IF (link status = lower) THEN
                {Check the clock of shorter time slots
                Allocate the time slot & assign corresponding λ
                Generate the control packet for the reservation of the resources
                Transmit the DB
}
END IF
        IF (link status = higher) THEN
                 {Check the clock of larger time slots
        Allocate the time slot & assign corresponding λ
        Generate the control packet for the reservation of the resources
        Transmit the DB
}
END IF
END
```

3.6 Flowchart of the proposed solution

Our proposed algorithm is based on a concept in which nodes are pre-programmed for separation of the shorter links and the longer links. When data packets get aggregated at the OBS node for a specific destination, clock is checked initially on the basis of link status, and the respective slot is assigned. After the slot allocation, wavelength λ is assigned to a data burst on the basis of modulo arithmetic table and control packet is generated for channel reservation. Finally, DB is transmitted after an offset. The flow chart of the algorithm is shown flowchart 1.


Data Burst (DB) aggregated
Check the corresponding clock
Slot allocation depending on the link
Assign λ according to clock
Generate the control packet for the reservation of resources
Transmit DB
Exit Scheduling

Flowchart.1: Flowchart of the proposed solution

4 Network Simulation modeling

We have implemented the proposed work into computer environment by using JAVA programming language. Object oriented approach is used for the implementation of the work by JAVA programming language.

4.1 Network Model

The proposed algorithm is designed for the OBS network, as depicted in figure 4.1. Access networks are connected with the edge nodes of the OBS network, data packets are coming from access network to the OBS network at edge nodes. The edge nodes are capable of assembling data packets into data bursts and deliver to the core nodes of the OBS network. All core nodes are capable to receive and transmit data bursts on different wavelengths at the same time using modulo arithmetic to avoid contention, as all core nodes are synchronized.

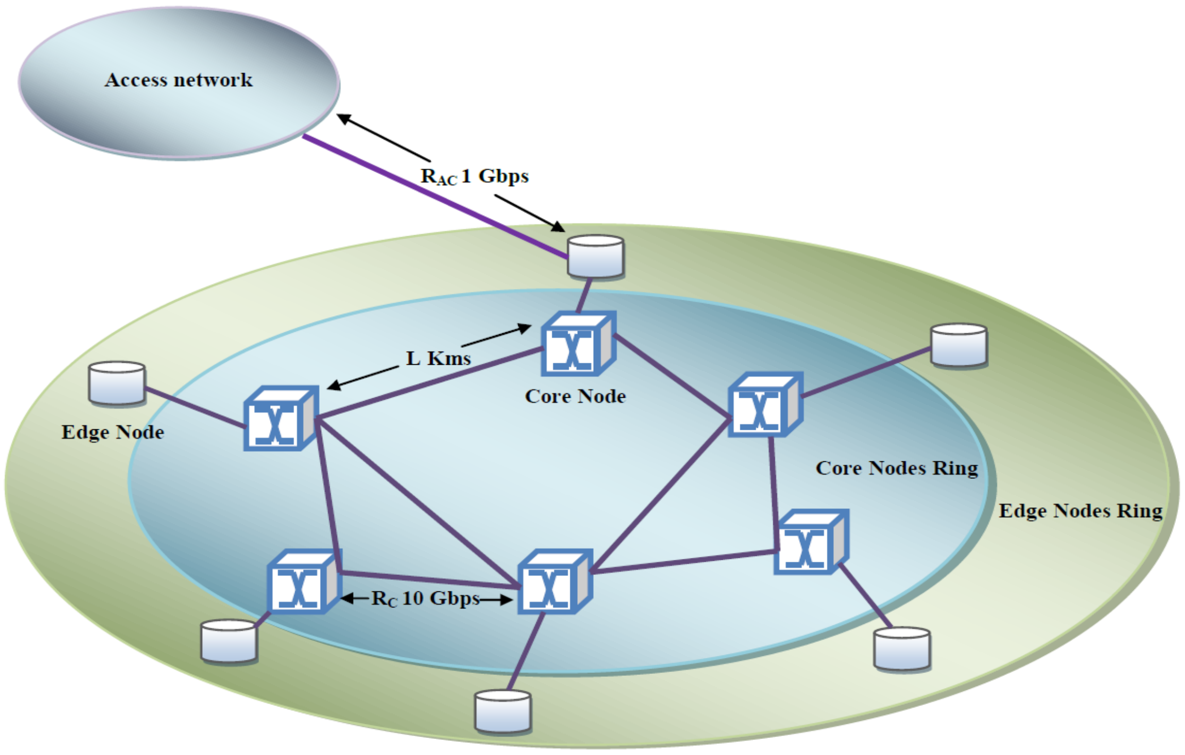


Figure 4.1: Proposed network architecture simulation model

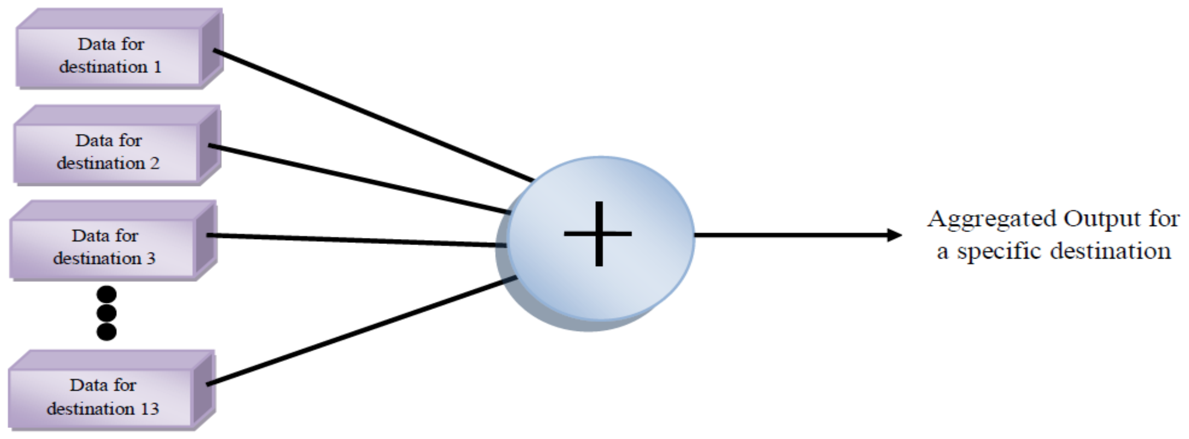
Figure 4.1 shows a set of core nodes in a core node ring which are connected to each other at 10 Gbps link at a distance ***L*** Km from each other (based on the actual network topology). Edge nodes are connected to the core nodes and to access networks over the link of 1 Gbps.

4.2 Algorithm

The proposed algorithm is designed for the OBS network. The edge node is capable of aggregating data packets from the access networks and form a data burst, the size of the data burst depends on the transmission delay of this specific data burst from source to destination. At the core node, destination of the data burst is analyzed and on the basis of the destination a specific time slot is assigned with an affiliated wavelength. These core nodes are synchronized and have same number of wavelengths, the wavelength assignment to data bursts is based on the wavelength allocation table which is derived from the modulo arithmetic

4.3 Traffic modeling

In our model data packets arrive at arbitrary edge node according to the Poisson distribution process with rate $\boldsymbol{\lambda}$. Data packets for the same destination gets aggregated to form a single data burst to be transmitted at the edge node. The total traffic load is varied at different rates (λ) from 0.25 to 3.5.



$$f(k;\mu) = \frac{\mu^{k}\, e^{-\lambda}}{k!} \qquad (5)$$

Where;

- e; base of natural logarithm (e = 2.71828…)
- k; number of occurrence of an event
- μ; positive real number, equal to the expected number of occurrences during the given interval (time)

Figure 4.2: Traffic generation model

Equation 5, describes the basic outline of Poisson process for generating ingress traffic to the given network model, ***k*** is a random number which describes occurrence of an event. Poisson distribution mean value, $\boldsymbol{\mu}$ represents the data traffic intensity for network model, we have simulated our solution at different traffic intensities $\boldsymbol{\mu}$ (0.25 to 3.5) to analyze the data packet loss. This data forms aggregated data to be transmitted to a specific destination inside the core network. The size of each data burst generated in variable, as we have adopted time based aggregation technique for data burst aggregation.

4.4 Simulation setup designing

We have implemented our proposed solution using computer simulation. In figure 4.3 simulation setup design is described. Traffic generator module is generating data traffic for random destinations using Poisson distribution. This data traffic is transported to the edge node by using Link module which transport at specific data rate. The edge node classifies the data traffic for each destination and holds in to holding buffer which are defined for each destination. The holding buffer transfers data packet to sending buffer where data burst aggregation takes place. The data burst aggregation time depends on the slot size for specific destination. At edge node if buffer is full, data packets will be dropped. The dropped data packets are stored. The edge node delivers the data burst to the core node where core node check the data burst core node destination and transmit the data burst using Link module at specific data rate inside the core network. The transmission of data burst involves scheduling a specific wavelength for the establishment of the light path. This scheduling is done on the basis of the algorithm, which is implemented in the core node module. If at any instance of time more than one core node assigns the same wavelength for the establishment of the light path, our setup indicates wavelength contention which is considered data burst dropping.

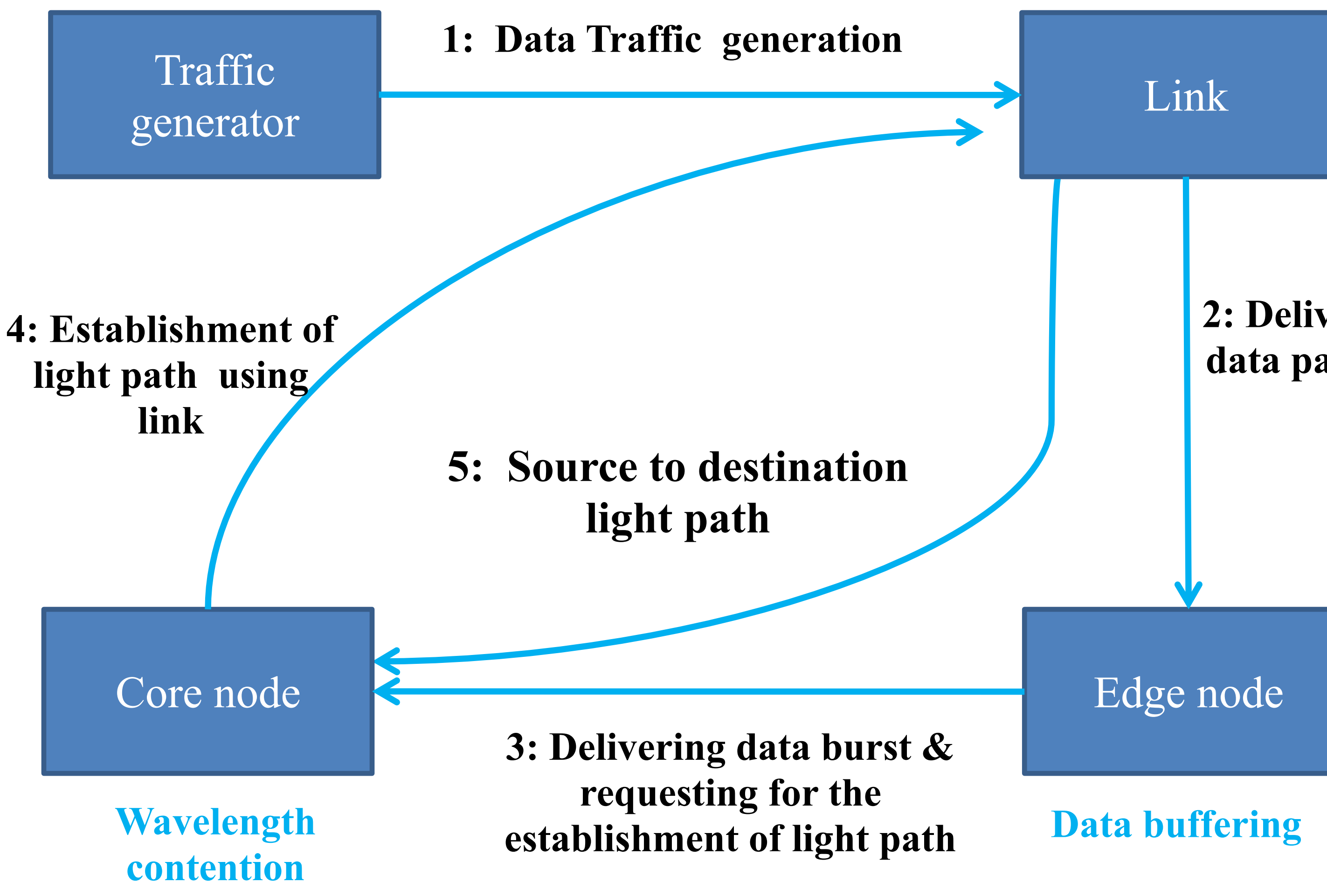

Traffic generator
1: Data Traffic generation
Link
4: Establishment of light path using link
2: Deliv
data pa
5: Source to destination light path
Core node
Edge node
3: Delivering data burst & requesting for the establishment of light path
Wavelength contention
Data buffering

Figure 4.3: Simulation setup design

Figure 4.4 illustrates the sequence diagram of simulation setup. Sequence diagram shows the sequence of events and flow of messages between the classes of the simulation setup. This illustration depicts how events are generated and how classes interact with each other while completing the simulation cycle. The detailed description of operations attributes and associations of classes are given below.

Description

Associations:

Buffer space:	Defines the limited space for the storage of data units coming to the node
Buffer limit:	Defines the limit after which data will be considered drop from the node
Buffer overflow:	Defines the number of data units dropped
Aggregation time:	Defines the time for which data units will be aggregated into a single data burst
Time slot:	Defines the time for which data burst will be delivered from source to destination
Source add:	Defines the source from which the data burst will be originated
Destination add:	Defines the destination where data will be delivered
Burst size:	Defines the aggregated data burst size
Wavelength:	Defines the wavelength which is allotted from source to destination establishing light path

Interfaces

Operations

Get_wavelength	Wavelength assigned for a specific light path
Get_BurstSize	Gives the size of data burst
Light_Path_estb	Gives the status whether light path is established or not
Link_rate	Gives the link rates in Mbps
Wavelength	Wavelength contention identifier

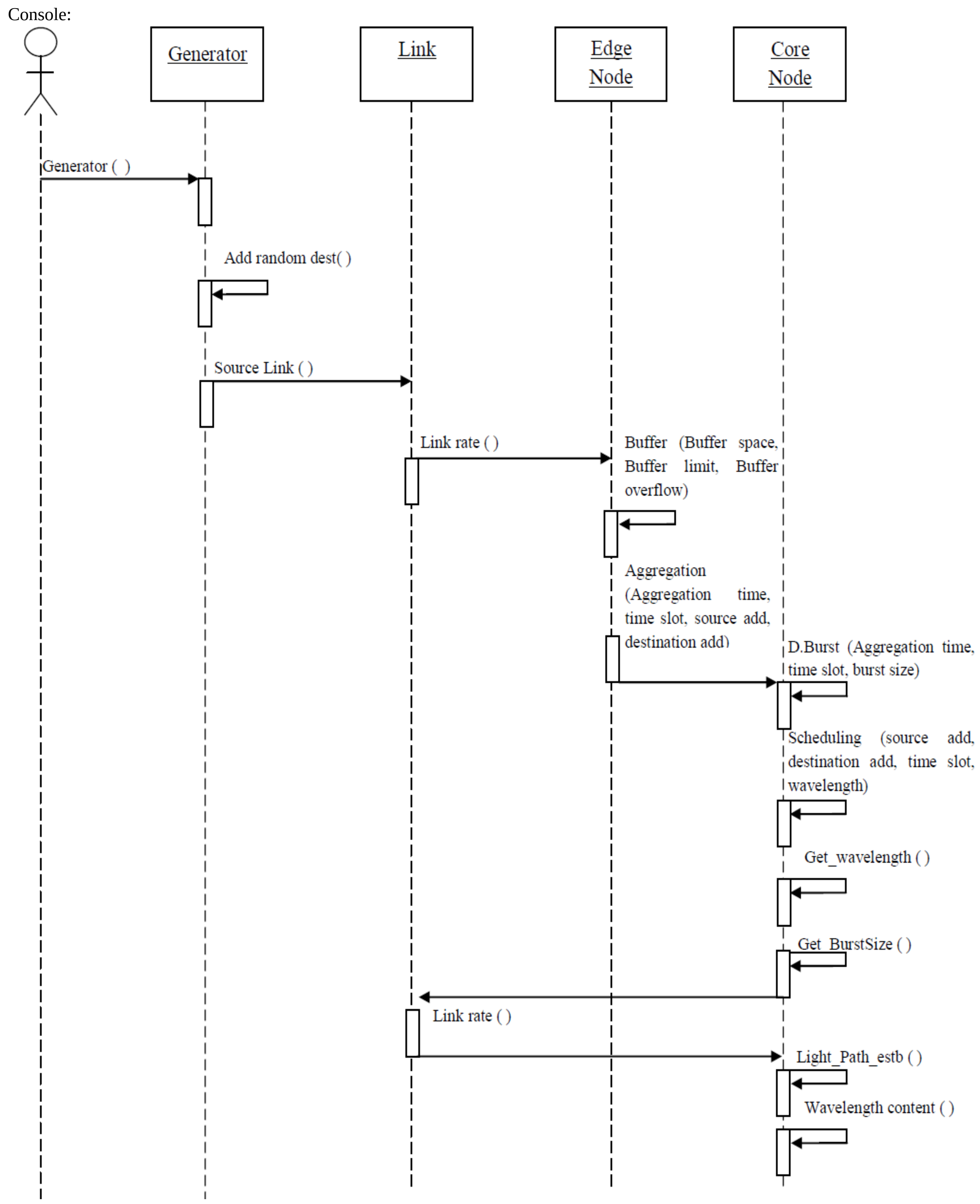

Console:
Generator
Link
Edge Node
Core Node
Generator ( )
Add random dest( )
Source Link ( )
Link rate ( )
Buffer (Buffer space, Buffer limit, Buffer overflow)
Aggregation (Aggregation time, time slot, source add, destination add)
D.Burst (Aggregation time, time slot, burst size)
Scheduling (source add, destination add, time slot, wavelength)
Get_wavelength ( )
Get_BurstSize ( )
Link rate ( )
Light_Path_estb ( )
Wavelength content ( )

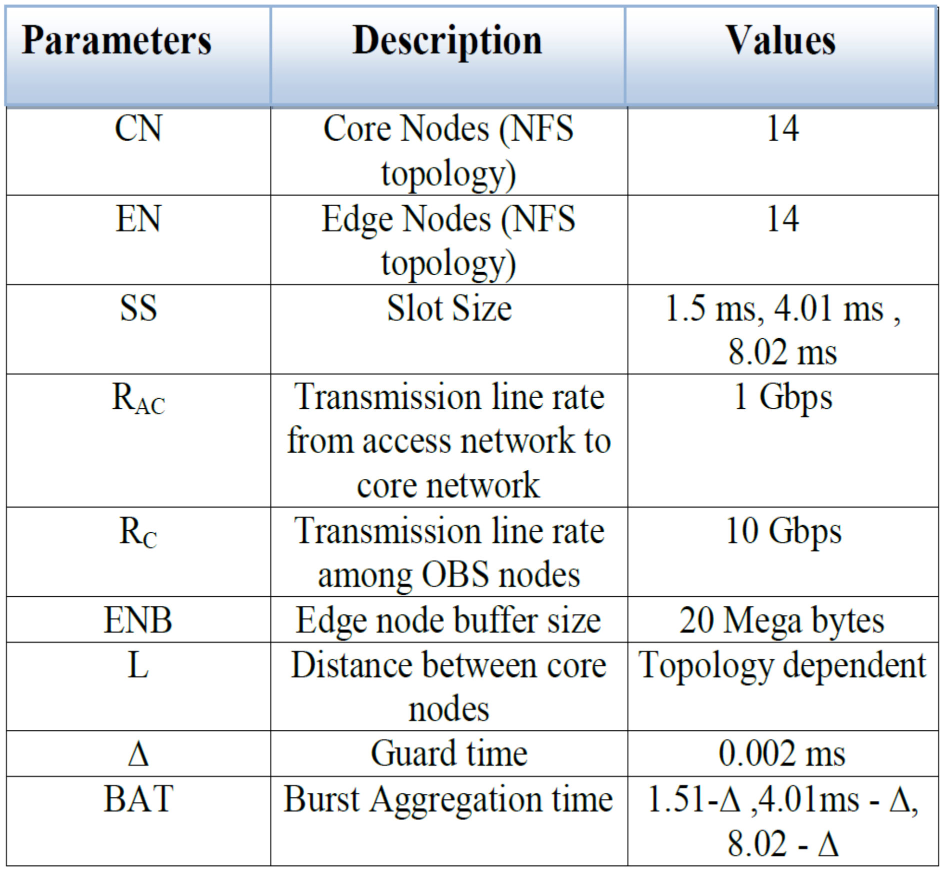
Figure 4.4: Sequence diagram of the simulation setup

4.5 Performance Evaluation

The system model was setup in figure 4.1 with edge nodes of the OBS network are connected to the access network. The transmission speed from the access network to the edge node is $\boldsymbol{R_{AC}}$ Gbps and the transmission speed among nodes of the OBS network is $\boldsymbol{R_{CC}}$ Gbps. Each edge node buffer ***ENB*** has a finite size and the distance between OBS is topology dependent is ***L*** Kms.

In our model, we used timer based aggregation [33] for the formation of data burst at the edge node. Data from the access network coming to the OBS network at the edge node, data burst is formed at burst aggregation time ***BAT*** which based on the Slot Size SS. The size of edge node buffer is taken 20 MB [28].

The following table shows the parameters used in our simulation.



Parameters	Description	Values
CN	Core Nodes (NFS topology)	14
EN	Edge Nodes (NFS topology)	14
SS	Slot Size	1.5 ms, 4.01 ms , 8.02 ms
R_{AC}	Transmission line rate from access network to core network	1 Gbps
R_{C}	Transmission line rate among OBS nodes	10 Gbps
ENB	Edge node buffer size	20 Mega bytes
L	Distance between core nodes	Topology dependent
Δ	Guard time	0.002 ms
BAT	Burst Aggregation time	1.51-Δ ,4.01ms - Δ, 8.02 - Δ

Table.4: Simulation System parameters

5 Simulation Results

Firstly in figure 5.2, the impact of traffic rate μ in our proposed solution is discussed and compared with the results reported in [28]. In [28], burst transmission algorithm is proposed for contention free environment in TSOBS. Two thresholds are utilized for the data burst transmission, request threshold ***C***' and assemble threshold ***C***. Request threshold is utilized to send request to the controller (upper plane of TSOBS) when number of data packets in the edge node buffer becomes equal to request threshold. If the number of data packets in the buffer becomes equal to the assembly threshold before the acknowledgement is received to the node from the controller, the node assembly data packets and transmit the data burst at the start of the assigned time slot. Otherwise, if the node receives the acknowledgement from the controller before the data packets reaching to the assembly threshold, the node assemble and transmit the burst at the start of the assigned time slot.

This proposed burst transmission algorithm lowers the queue delay of data packets at the buffers of the edge nodes. As in other previously proposed burst transmission algorithms, the edge node solely depends on the acknowledgement of the controller [28].

In this paper the authors have investigated that how their proposed algorithm performs at different value of request threshold ***C***' and assemble threshold ***C***. In the figure 5.1, data packets loss probability is investigated against the data packet arrival rate. When the data packets loss probability is smaller than 0.01, the proposed solution at ***C***' =4000 can decrease the loss probability by over 90% [28].

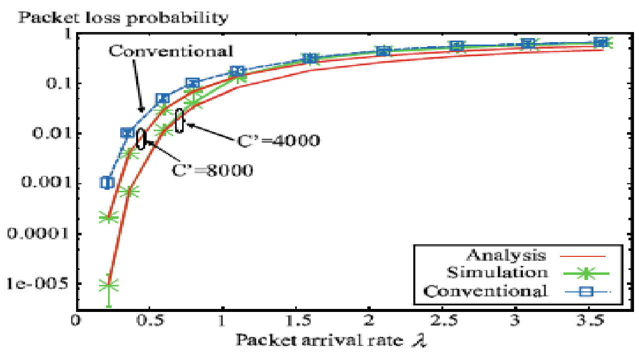


Figure 5.1: data packets loss probability against packets arrival rate [28]

In figure 5.2 network trend in terms of data loss probability is presented against the different data packets arrival rate. Lower data loss probability has been observed in case of our proposed solution, figure 5.2 (square line graph), as compared to the solution reported in [28] (circle line graph), at slot size of 7.5 ms.

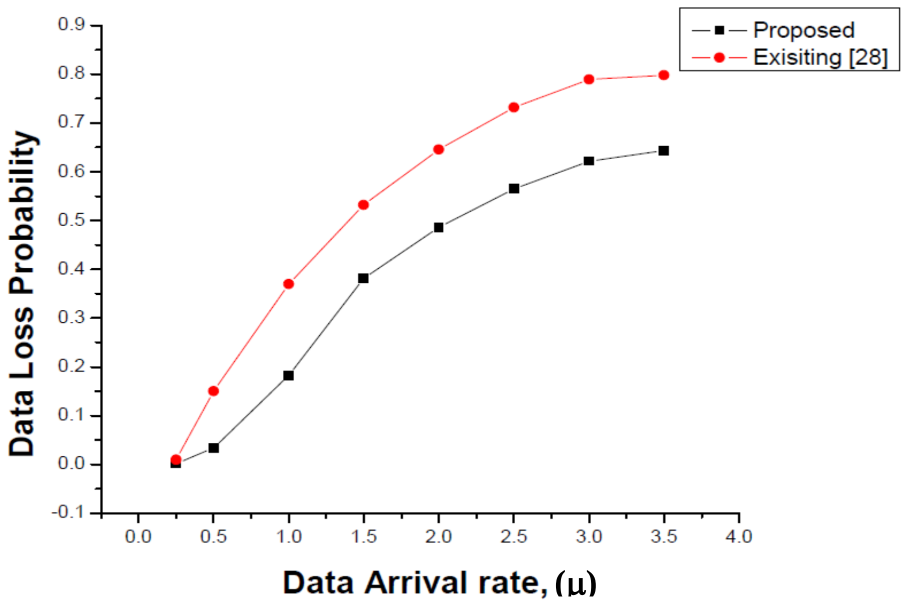


Figure 5.2: Data packets arrival rate against data loss probability

The proposed solution does not request for the slot allotment for a data burst which results in lower queuing delay for the data packets in the buffer queue. However, the data loss probability increases for high data arrival rates.

Data packets arrival rate is the number of data packets arriving at OBS node from the access network. The channel bandwidth utilized in actual network is around 100 – 200 Mbps. The data arrival rates of interest in terms of mean value of Poisson distribution is around 1μ. In figure 5.3, the data packets arrival rate is plotted against the mean bandwidth utilized by the channel in sending the data packets to the network node. As the data packet arrival rate is increasing the channel is utilization (bandwidth) is also increasing and more data traffic is arriving to the network.

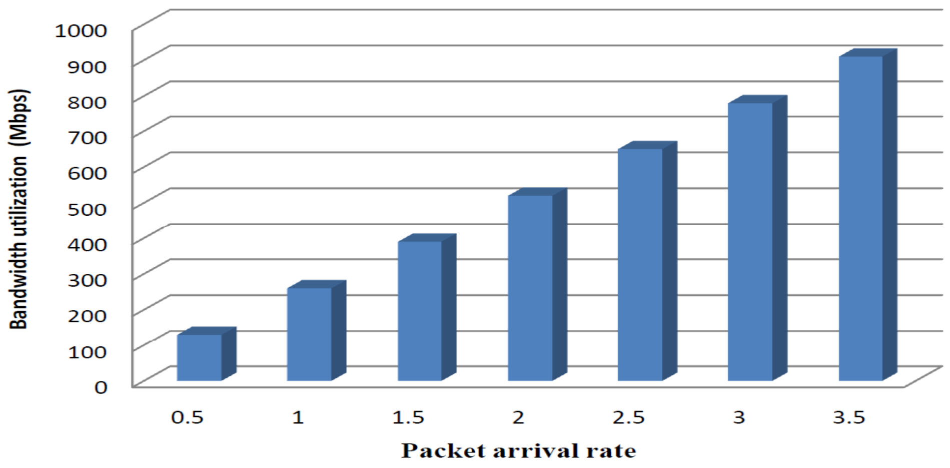


Figure 5.3: Packets arrival rate against mean channel bandwidth utilization (Mbps)

The data packets arrival rate of interest is around 1 (mean value of Poisson distribution) and figure.5.3 illustrate the number of data packets at different data arrival rate. In figure 5.4, the number of data packets is plotted against the data loss probability, here the performance results of our proposed solution is compared with existing proposed solution in [28]. The proposed solution is showing better data loss probability, at data arrival rate of μ=0.3 our proposed solution is showing 68% better performance.

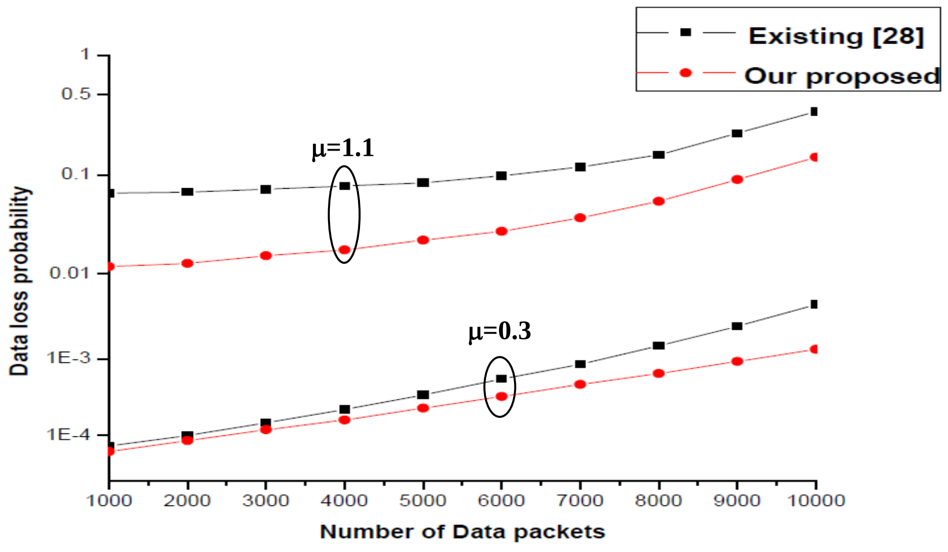


Figure 5.4: Number of Data packets against the data loss probability

In figure 5.5 data loss probability has been given at two different slot sizes. Data loss probability increases with the increase in the slot size, as for the bigger slot size more data will have to wait in the buffer which results in early overflow of the buffer.

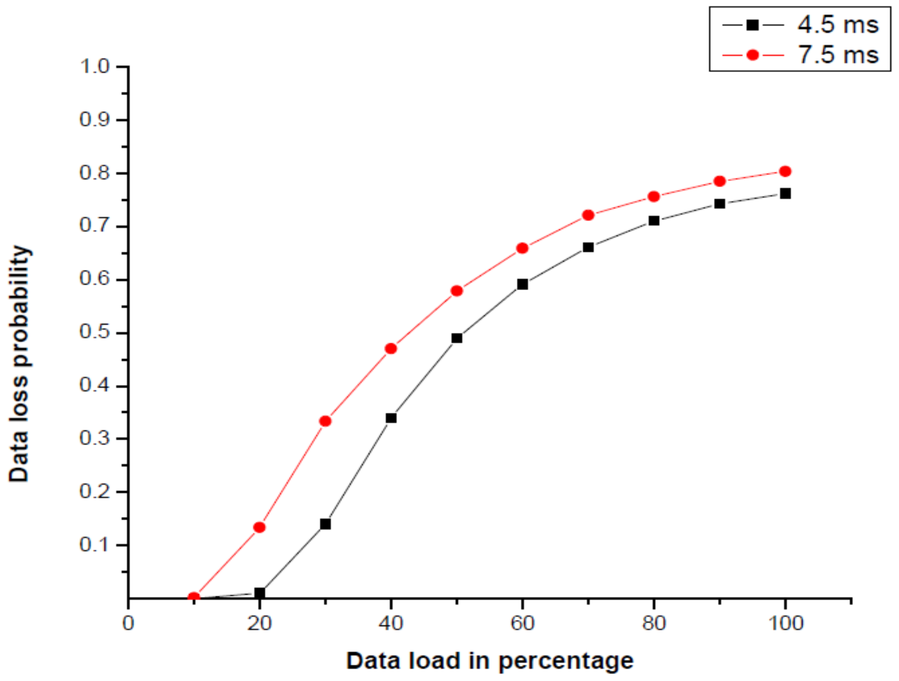


Figure 5.5: Data load in percentage against data loss rate (for two different slot sizes 4ms & 8ms)

In figure 5.6, slot sizes are plotted against the total number of wavelengths required to ensure wavelength contention free environment in an OBS network. Number of wavelengths is strictly dependant on the modulo table (slot allocation table) construction for derived number of classes of transmission delays. For smaller slot size larger number of classes of transmission delay will be formed which results in larger wavelength requirement for contention free environment. Larger number of slot size usage results in less number of wavelength requirement.

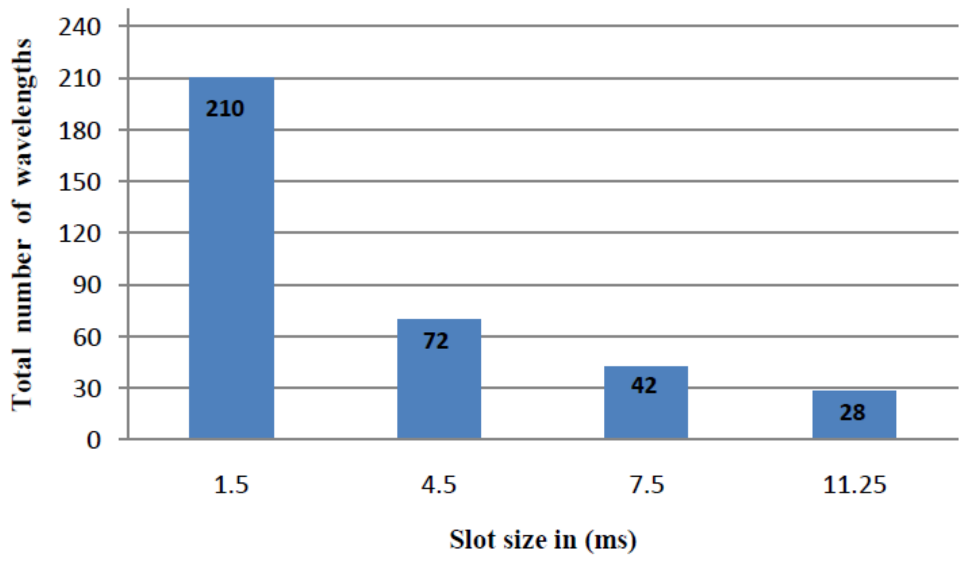


Figure 5.6: Total number of wavelengths required against Slot size (ms)

Simulated results show that if larger value of slot size is considered than larger data loss is observed as compared to smaller slot size. Similarly if smaller slot size is considered there is need of larger number of wavelength required to avoid wavelength contention as compared to larger value of slot size. On the basis of the inference of simulated results the optimum slot size is 7.5 ms which yields better results against data loss probability and wavelength requirement.

6 Conclusion & Future directions

This thesis concludes with proposed improved network architecture of time slotted OBS network and delay aware burst transmission algorithm which is aimed to minimize data burst loss and improves packet level performance in OBS network. The improved TS-OBS network reduces extra delay of the data packets at edge node buffer, which is incurred while node request for the allotment of time slots and waiting for positive acknowledgement for data burst transmission. This proposed network architecture does not request for the allotment of time slots for data transmission instead checks the corresponding clock and transmit data burst.

We have provided contention free environment in this network architecture by the formation of slot allocation table using modulo arithmetic. Slot allocation table allocates a time slot with an associated wavelength in such a way that it avoids wavelength contention. The transmission delays of all possible data burst links are divided into multiple classes. The division of these classes depends on the highest transmission delay in the network topology and the slot size. The slot sizes of classes should be the complete divisor to the highest transmission delay of the network link. The proposed delay aware data burst transmission algorithm assigns different time slots to each data burst depending upon transmission delay class, which improves packet level performance.

Calculated results show that our proposed solution provides improved performance over the earlier solutions; our simulation setup indicates that at least 67% improvement in lowering data packet loss probability has been achieved at 0.3 data arrival rate (Poisson distribution mean value) than the existing burst transmission approach proposed by Hozumi Kawanami in Journal: Photonic Network

Communication (2010). Results inference suggests that 7.5 ms is the optimum slot size which yields comparatively better results against data loss probability and total number of wavelengths required in contention free environment.

Our proposed solution is effective in DWDM network environments, where large number of wavelengths are available. Large number of wavelengths will increases the number of transmission delay classes, thus reducing queuing delay of links to larger extent.

The proposed solution can further be analyzed by the introduction of optical buffers at the core nodes. Combining proposed solution with the optical buffers can reduce the data load at the edge nodes which might result in better network performance. Wavelength constraint on the proposed solution may yield better results in terms of better network utilization and data loss. These could be the potential future directions of the proposed work.

7 References

[1] Ujager, F.S.; Zaidi, S.M.H.; Younis, U.; *"A review of semiconductor lasers, present and future application in optical communication"* High-Capacity Optical Networks and Enabling Technologies (HONET), Publication Year: 2010 , Page(s): 274 – 279

[2] David F. Welch, *"A Brief History of High-Power Semiconductor Lasers*," IEEE Journal of selected topics in quantum electronics, Vol. 6, No. 6, 2000.

[3] Harry J. R. Dutton, *"IBM Understanding Optical Communication"* First Edition, Sep. 1998.

[4] E. Desurvire; M. Zirngibl; H.M. Presby; D. DiGiovanni *"Dynamic gain compensation in saturated erbium-doped fiber amplifiers"* Photonic Technology Letters, IEEE, 1991, pp :453-455

[5] J. P. Jue and V. M. Vokkarane, "*Optical burst switched networks",* Springer, Optical Networks Series, 2005.

[6] X. Huang, F. Farahmand, and J. P. Jue, "*An Algorithm for Traffic Grooming in WDM Mesh Networks with Dynamically Changing Light- Trees,"* Proceedings of the IEEE GLOBECOM'04, Dallas, USA, pp. 1813-1817, 29 November to 3 December 2004.

[7] G. K. Chang, J. Yu, Y. K. Yeo, A. Chowdhury, and Z. Jia, *"Enabling technologies for next-generation optical packet-switching networks,"* Proceedings of the IEEE, vol. 94, no. 5, pp. 892 – 910, May 2006

[8] V. R. Jonnadula, "*Performance Analysis of Congestion Control Schemes in OBS Mesh Networks,"* MSc. Thesis, Available: http://www4.ncsu.edu/~hp/VenkatThesis.pdf, (Last accessed 15 September 2011), NCSU, 2004.

[9] L. Battestilli, Harry G. Perros, *"A performance study of an optical burst switched network with dynamic simultaneous link possession,"* Computer Networks: The International Journal of Computer and Telecommunications Networking, vol.50 no.2, pp.219-236, 08 February 2006.

[10] 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, 1999, pp. 69–84.

[11] Lina and Harry Peros *" Optical Burst Switching: A survey"*

[12] Xenia Mountrouidou and Harry G. Perros *"Characterization of the Burst Aggregation Process in Optical Burst Switching"* Proceedings of international conference on Networks for grid applications, publishers: ICST, atcile no. 28, 2007

[13] Yu, X., Li, J., Cao, X., Chen, Y., Qiao, C.: *"Traffic statistics and performance evaluation in optical burst switched networks."* Journal of lightwave technology 22(12) (2004), pp 2722–2738

[14] M. Dueser and P. Bayvel *" Band width utilization and wavelength resuse in WDM optical burst switched packet networks*" Preecedings of IFIP 5th working conference on optical network design and modeling, 2001, pp 23-24.

[15] G. Hudek and D. Muder. "*Signaling analysis for a multi-switch all-optical network*". In Proceedings of IEEE International Conference on Communications-ICC'95, Seattle, June 1995,pp 1206-1210.

[16] C. Qiao and M . Yoo. "*Choices, features, and issues in optical burst switching."* Optical Network Magazine, April 2000, pp 36-44.

[17] J. Y . Wei and R. I. McFarland." *Just-in-time signaling for W D M optical burst switching networks." Journal* of Lightwave Technology, December 2000, pp 2019-2037.

[18] M . Yoo. M . Jeong, and C. Qiao. *"A high-speed protocol for bursty traffic in optical networks." In* SPIE Proceedings, All Optical Networking: Architecture, Control and Network Issues, November 1997, pp 79-90.

[19] J. Turner *" Terabit burst switching"* Journal of High Speed Network, 1999

[20] Ilia Baldin & Harry Peros *" Jumpstrat: Just in time signaling architecture for WDM burst switched network"* IEEE Communication magazine., 2003, pp 82-89.

[21] M. Yang, S.Q. Zheng, and D. Verchere, "*A qos supporting scheduling algorithm for optical burst switching in dwdm networks,"* Proceedings of GLOBECOM 2001, pp. 86–91, 2001.

[22] C. Murthy and M. Gurusamy, "*WDM Optical Networks: Concepts, Design, and Algorithms"*, Prentice Hall, 2002.

[23] J. Ramamirtham and J. Turner, "*Time Sliced Optical Burst Switching,"* in Proc. Infocom, 2003, pp. 2030–2038

[24] Onur Ozturk, Ezhan Karasan and Nail Akar *"Performance Evaluation of Slotted Optical Burst Switching Systems With Quality of Service Differentiation"* JOURNAL OF LIGHTWAVE TECHNOLOGY, VOL. 27, NO. 14, JULY 15, 2009, pp- 2621-2634

[25] Tai-Won Um, Jun Kyun Choi, Seong Gon Choi, Won Ryu *"Performance Analysis of a Centralized Resource Allocation Mechanism for Time-Slotted OBS Networks"* International Conference on Networking and Services (2006) (LNCS: Volume 4238/2006, pp 403-411)

[26] "Architecture for the automatically switched optical network (ASON)", ITU-T Recommendation, G.8080/Y.1304, Nov. 2001

[27] Abdeltouab Belbekkouche, Abdelhakim Hafid *"An Absolute And Fair QoS Differentiation Scheme For DWDM OBS Networks"* , Global telecommunication conference 2009, IEEE page 1 to 7

[28] Hozumi Kawanami , Hiroyuki Masuyama , Takuji Tachibana, Shoji Kasahara & Yutaka Takahashi *"Burst transmission algorithm to improve packet level performance in contention-free slotted OBS networks"* Journal: Photonic Network Communication (2010) pp:54–63 (Springer)

[29] Y. Chen, Jonathan S. Turner & Zhi Zhai *"Contour-Based Priority Scheduling in Optical Burst Switched Networks"* Journal of Lightwave Technology, Vol. 25, No. 8, Aug 2007

[30] Yonggyu Lee · Jaegwan Kim · Minho Kang, *"Feasibility analysis for service differentiation using an FDL bank in OBS networks",* Photonic Network Communications, Volume 15, Number 3 / June, 2008. Pages 275-281

[31] Maurizio Casoni, Enrica Luppi and Maria Luisa Merani *"Impact of Assembly Algorithms on End-to-End Performance in Optical Burst Switched Networks with Different QoS Classes"* BROADNet 2004, IEEE Computer Society (2004)

[32] Shlomo Ovadia, Christian Maciocco, and Mario Paniccia *"Photonic burst switching (PBS) Architecture for Hop and Span-Constrained Optical Network"* Journal: IEEE Optical Communication, Volume: 41Issue 11, Nov (2003) pp: S24 - S32

[33] Nail Akar , Ezhan Karasan , Kyriakos G. Vlachos , Emmanouel A. *Varvarigos "A survey of quality of service differentiation mechanisms for optical burst switching networks"* Optical Switching and Networking 7 (2010), pp: 1-11

8 Appendix A: Poisson Distribution process

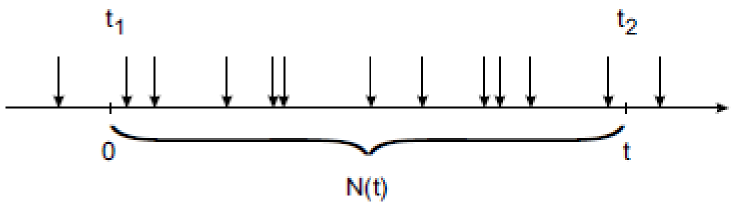
The text has been taken from the tutorial by J.Virtamo

General

Poisson process is one of the most important models used in queueing theory.

• Often the arrival process of customers can be described by a Poisson process.

• In teletraffic theory the “customers” may be calls or packets. Poisson process is a viable model when the calls or packets originate from a large population of independent users. In the following it is instructive to think that the Poisson process we consider represents discrete arrivals (of e.g. calls or packets).



Mathematically the process is described by the so called counter process Nt or N(t). The counter tells the number of arrivals that have occurred in the interval (0, t) or, more generally, in the interval (t1, t2).

$$\begin{cases} N(t) = \text{number of arrivals in the interval } (0,t) & \text{(the stochastic process we consider)} \\ N(t_1,t_2) = \text{number of arrival in the interval } (t_1,t_2) & \text{(the increment process } N(t_2)-N(t_1)\text{)} \end{cases}$$

A Poisson process can be characterized in different ways:

• Process of independent increments

• Pure birth process

– The arrival intensity λ (mean arrival rate; probability of arrival per time unit

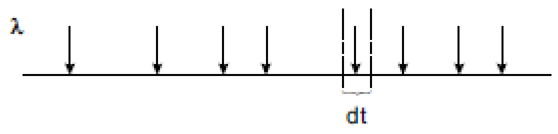
• The “most random” process with a given intensity λ

Definition

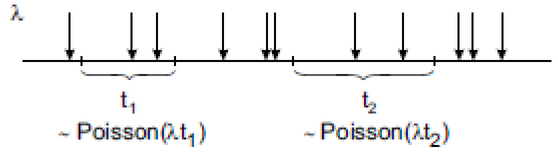
The Poisson process can be defined in three different (but equivalent) ways:

1. Poisson process is a pure birth process:

In an infinitesimal time interval *dt* there may occur only one arrival. This happens with the probability λdt independent of arrivals outside the interval.

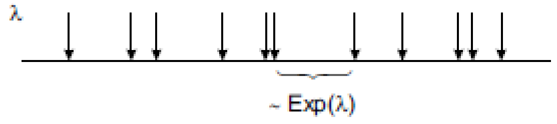


2.The number of arrivals N(t) in a finite interval of length t obeys the Poisson(λt) distribution, Moreover, the number of arrivals N(t1, t2) and N(t3, t4) in non-overlapping intervals (t1 ≤ t2 ≤ t3 ≤ t4) are independent.



3. The inter arrival times are independent and obey the Exp(λ) distribution:

$$P\{\text{interarrival time} > t\} = e^{-\lambda t}$$



Properties of the Poisson process

The Poisson process has several interesting (and useful) properties:

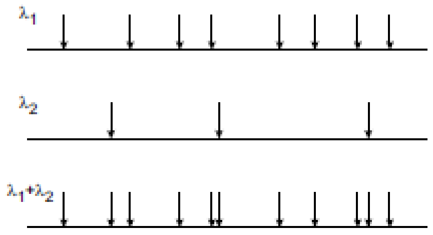
1. **Conditioning on the number of arrivals.** Given that in the interval (0, t) the number of arrivals is N(t) = n, these n arrivals are independently and uniformly distributed in the interval.

• One way to generate a Poisson process in the interval (0, t) is as follows:

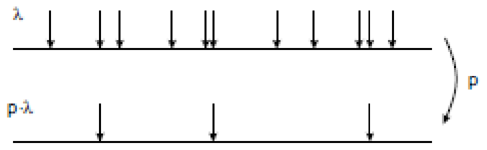
– draw the total number of arrivals n from the Poisson(λt) distribution

– for each arrival draw its position in the interval (0, t) from the uniform distribution, independently of the others

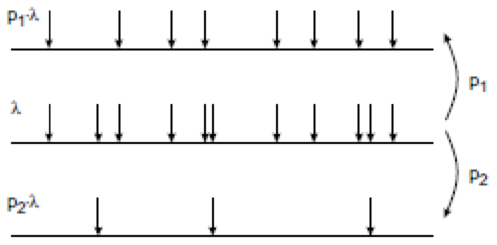
2. **Superposition.** The superposition of two Poisson processes with intensities λ1 and λ 2 is a Poisson process with intensity λ = λ1 + λ2.



3. **Random selection.** If a random selection is made from a Poisson process with intensity λ such that each arrival is selected with probability p, independently of the others, the resulting process is a Poisson process with intensity pλ.



4. **Random split.** If a Poisson process with intensity λ is randomly split into two subprocesses with probabilities p1 and p2, where p1 + p2 = 1, then the resulting processes are independent Poisson processes with intensities p1 λ ja p2 λ. (This result allows an straight forward generalization to a split into more than two sub processes.)



5. **PASTA.** The Poisson process has the so called PASTA property (Poisson Arrivals See Time Averages): for instance, customers with Poisson arrivals see the system as if they came into the system at a random instant of time (despite they induce the evolution of the system).

9 Appendix B: Simulation Source Code

```
import java.util.Random;

import java.lang.Math;

class simulation {

public static void main(String args[]) {

int fact;

 double lemda1=0.5;double lemda2=1.0;double lemda3=1.5;double lemda4=2.0;double lemda5=2.5;
;double lemda6=3.0;double lemda7=3.5;

// different data arrival rates of  data packets

Int packet=1250;  // Data packet size

double neg1= -1*lemda1;  double neg2= -1*lemda2;double neg3= -1*lemda3;double neg4= -
1*lemda4;double neg5= -1*lemda5; double neg6= -1*lemda6;double neg7= -1*lemda7; double y1;double
y2;double y3;double y4;double y5; double y6;double y7;
double c1,c2,c3,c4,c5,c6,c7,c8,c9,c10,c11,c12,c13,d1,d2,d3,d4,d5,d6,d7, d8,d9,d10,d11,d12,d13;

// The above code lines showing the variables which are required for the Poisson distribution

double buffer1=20*1024;double buffer2=20*1024;double buffer3=20*1024;double
buffer4=20*1024;double buffer5=20*1024; double buffer6=20*1024;double buffer7=20*1024; double
buffer8=20*1024;double buffer9=20*1024;double buffer10=20*1024;double buffer11=20*1024; double
buffer12=20*1024;double buffer13=20*1024;

//Defining buffer sizes
                    int slot=11.25;              //slot size is defined
                    int accum=0;                 // dummy variable
int b;
for (int p=1;p<26000;p++){    // it is the counter value
int x = src1.nextInt(1);
b=fact(x);
c1=Math.pow(lemda1,x);c2=Math.pow(lemda2,x);c3=Math.pow(lemda3,x);c4=Math.pow(lemda4,x);c5=
Math.pow(lemda5,x);
c6=Math.pow(lemda6,x);c7=Math.pow(lemda7,x);c8=Math.pow(lemda8,x);c9=Math.pow(lemda9,x);c10
=Math.pow(lemda10,x);
c11=Math.pow(lemda11,x);c12=Math.pow(lemda12,x);c13=Math.pow(lemda13,x);
```

```
d1=Math.pow(2.718,neg1);d2=Math.pow(2.718,neg2);d3=Math.pow(2.718,neg3);d4=Math.pow(2.718,ne
g4);d5=Math.pow(2.718,neg5);
d6=Math.pow(2.718,neg6);d7=Math.pow(2.718,neg7);d8=Math.pow(2.718,neg8);d9=Math.pow(2.718,ne
g9);d10=Math.pow(2.718,neg10);
d11=Math.pow(2.718,neg11);d12=Math.pow(2.718,neg12);d13=Math.pow(2.718,neg13);

double e=b;
double f=1/e;

y1=c1*d1*f;y2=c2*d2*f;y3=c3*d3*f;y4=c4*d4*f;y5=c5*d5*f;
y6=c6*d6*f;y7=c7*d7*f;y8=c8*d8*f;y9=c9*d9*f;y10=c10*d10*f;
y11=c11*d11*f;y12=c12*d12*f;y13=c13*d13*f;y14=c14*d14*f;y15=c15*d15*f;
y1=y1*packet; y2=y2*packet;  y3=y3*packet; y4=y4*packet; y5=y5*packet; y6=y6*packet;
y7=y7*packet; y8=y8*packet; y9=y9*packet; y10=y10*packet; y11=y11*packet; y12=y12*packet;
y13=y13*packet; y14=y14*packet;

double accum1=0;double accum2=0;double accum3=0;double accum4=0;double accum5=0; double
accum6=0;double accum7=0;double accum8=0;double accum9=0;double accum10=0; double
accum11=0;double accum12=0;double accum13=0;
 accum1= y1+accum1;accum2= y2+accum2; accum3= y3+accum3; accum4= y4+accum4; accum5=
y5+accum5; accum6= y6+accum6;accum7= y7+accum7; accum8= y8+accum8; accum9= y9+accum9;
accum10= y10+accum10; accum11= y11+accum11;accum12= y12+accum12; accum13= y13+accum13;
if (p==slot){ buffer1=buffer1-accum1;buffer2=buffer2-accum2;buffer3=buffer3-
accum3;buffer4=buffer4-accum4;buffer5=buffer5-accum5; buffer6=buffer6-accum6;buffer7=buffer7-
accum7;buffer8=buffer8-accum8;buffer9=buffer9-accum9;buffer10=buffer10-
accum10;accum=accum+slot;}
if (p-accum==slot){ accum=accum+slot; buffer1=buffer1-accum1;buffer2=buffer2-
accum2;buffer3=buffer3-accum3;buffer4=buffer4-accum4;buffer5=buffer5-accum5; buffer6=buffer6-
accum6;buffer7=buffer7-accum7;buffer8=buffer8-accum8;buffer9=buffer9-accum9;buffer10=buffer10-
accum10; buffer11=buffer11-accum11;buffer12=buffer12-accum12;buffer13=buffer13-accum13; }
System.out.print(buffer1+"        "+buffer2+"              "+buffer3+"              "+buffer4+"
        "+buffer5+"     "+buffer6+"              "+buffer7+"              "+buffer8+"
        "+buffer9+"     "+buffer10+"             "+buffer11+"             "+buffer12+"
        "+buffer13);
// buffer data drop output
// The above code lines are showing the implementation of Poisson distribution

//  Bandwidth utilization using link of 1 Gbps

Int total_packet= p* packet*1024*1024*8; // representing data in bps
Int bw1 = total_packet * lemda1; Int bw2 = total_packet * lemda2; Int bw3 = total_packet * lemda5; Int
bw4 = total_packet * lemda4; Int bw5 = total_packet * lemda5; Int bw6 = total_packet * lemda6; Int bw7
= total_packet * lemda7; // for every data arrival rate
```

```
System.out.print(bw1+"              "+ bw2+"                      "+ bw3+"                      "+ bw4+"
             "+ bw5+"           "+ bw6+"                      "+ bw7);

static int fact(int n) {

             // Base Case:
             //    If n <= 1 then n! = 1.
             if (n <= 1) {
                 return 1;
             }
             // Recursive Case:
             //    If n > 1 then n! = n * (n-1)!
             else {
                 return n * fact(n-1);
             }
}}
// this section is checking wavelength contention

int n1,n2,n3,n4,n5,n6,n7,n8,n9,n10,n11,n12,n13,n14;

// 14 nodes are defined for NFSNet topology

int b1=0; int b2=0; int b3=0; int b4=0; int b5=0; int b6=0; int b7=0; int b8=0; int b9=0; int b10=0; int
b11=0; int b12=0; int b13=0; int b14=0;

// wavelength contention  initially all are zero

int
count1,count2,count3,count4,count5,count6,count7,count8,count9,count10,count11,count12,count13,cou
nt14;
//counters are defined to know the number of wavelength contention
int bl=0; {
                          for(i=0; i<=p; i++){
                          int lemda1[] =
{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28};
//28 wavelengths are defined depending on the slot siz i.e 11.25


                          Random src1 = new Random();
                          int num;
                          b1=0;
                          for (count=1; count<=p; count++) {
                          num = 1+src1.nextInt(4);
                          switch(num) {
```

```
// the condition below is checking all links for wavelength contention in 14 nodes nsfnet topology,
shortest path is implemented
        case 1:
                break;
                case 2:
                n1 = lemda1[i];
                n2 = lemda1[i];
                b1=b1+1;
        break;
                case 3:
                n1 = lemda1[i];
                n3 = lemda1[i];
                b1=b1+1;
                break;
                case 4:
                n1 = lemda1[i];
                n2 = lemda1[i];
                n4 = lemda1[i];
                b1=b1+1;
                break;

                case 5:
                n1 = lemda1[i];
                n2 = lemda1[i];
                n4 = lemda1[i];
                n5 = lemda1[i];
                b1=b1+1;
                break;

                case 6:
                n1 = lemda1[i];
                n2 = lemda1[i];
                n3 = lemda1[i];
                n6 = lemda1[i];
                b1=b1+1;
                break;

                case 7:
                n1 = lemda1[i];
                n2 = lemda1[i];
                n4 = lemda1[i];
                n5 = lemda1[i];
                n7 = lemda1[i];
```

```
b1=b1+1;
break;

case 8:
n1 = lemda1[i];
n8 = lemda1[i];
b1=b1+1;
break;

case 9:
n1 = lemda1[i];
n8 = lemda1[i];
n9 = lemda1[i];
b1=b1+1;
break;

case 10:
n1 = lemda1[i];
n8 = lemda1[i];
n9 = lemda1[i];
n10 = lemda1[i];
b1=b1+1;
break;

case 11:
n1 = lemda1[i];
n2 = lemda1[i];
n4 = lemda1[i];
n11 = lemda1[i];
b1=b1+1;
break;

case 12:
n1 = lemda1[i];
n8 = lemda1[i];
n9 = lemda1[i];
n12 = lemda1[i];
b1=b1+1;
break;

case 13:
n1 = lemda1[i];
```

```
                    n8 = lemda1[i];
                    n9 = lemda1[i];
                    n14 = lemda1[i];
                    n13 = lemda1[i];
                    b1=b1+1;
                    break;

                    case 14:
                    n1 = lemda1[i];
                    n8 = lemda1[i];
                    n9 = lemda1[i];
                    n14 = lemda1[i];
                    b1=b1+1;
                    break;
}}

int lemda2[] = {3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28AA,1,2};

                    Random src2 = new Random();
                    int num2;
                    b2=0;

                    for (count2=1; count2<=14; count2++) {
                    num2 = 1+src2.nextInt(14);
                    switch(num2) {

                    case 1:
                    n1 = lemda2[i];
                    n2 = lemda2[i];
                    b2++;
                    break;

                    case 2:


                    break;

                    case 3:
                    n2 = lemda2[i];
                    n3 = lemda2[i];
                    b2++;
                    break;

                    case 4:
```

```
n2 = lemda2[i];
n4 = lemda2[i];
b2++;
break;

case 5:
n2 = lemda2[i];
n5 = lemda2[i];
n4 = lemda2[i];

b2++;
break;

case 6:
n2 = lemda2[i];
n3 = lemda2[i];

n6 = lemda2[i];
b2++;
break;

case 7:
n2 = lemda2[i];
n4 = lemda2[i];
n5 = lemda2[i];
n7 = lemda2[i];


b2++;
break;

case 8:
n2 = lemda2[i];
n4 = lemda2[i];
n5 = lemda2[i];
n7 = lemda2[i];
n8 = lemda2[i];

b2++;
break;

case 9:
n2 = lemda2[i];
```

```
n4 = lemda2[i];
n5 = lemda2[i];
n7 = lemda2[i];
n8 = lemda2[i];
n9 = lemda2[i];

b2++;
break;

case 10:
n2 = lemda2[i];
n3 = lemda2[i];
n6 = lemda2[i];
n10 = lemda2[i];
b2++;
break;

case 11:
n2 = lemda2[i];
n4 = lemda2[i];
n11 = lemda2[i];

b2++;
break;

case 12:
n2 = lemda2[i];
n4 = lemda2[i];
n11 = lemda2[i];
n12 = lemda2[i];
b2++;
break;

case 13:
n2 = lemda2[i];
n4 = lemda2[i];
n11 = lemda2[i];
n12 = lemda2[i];
n13 = lemda2[i];
b2++;
break;

case 14:
n2 = lemda2[i];
```

```
                    n4 = lemda2[i];
                    n5 = lemda2[i];
                    n7 = lemda2[i];
                    n8 = lemda2[i];
                    n9 = lemda2[i];
                    n14 = lemda2[i];
                    b2++;
                    break;
}}

int lemda3[] = {5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,1,2,3,4};

                    Random src3 = new Random();
                    int num3;
                    b3=0;
                    for (count3=1; count3<=14; count3++) {
                    num3 = 1+src3.nextInt(14);
                    switch(num3) {

                    case 1:
                    n1 = lemda3[i];
                    n3 = lemda3[i];
                    b3++;
                    break;

                    case 2:
                    n2 = lemda3[i];
                    n3 = lemda3[i];
                    b3++;
                    break;

                    case 3:

                    break;

                    case 4:

                    n3 = lemda3[i];
                    n2 = lemda3[i];
                    n4 = lemda3[i];
                    b3++;
                    break;

                    case 5:
```

```
n3 = lemda3[i];
n2 = lemda3[i];
n5 = lemda3[i];
n4 = lemda3[i];

b3++;
break;

case 6:
n6 = lemda3[i];
n3 = lemda3[i];

b3++;
break;

case 7:
n2 = lemda3[i];
n3 = lemda3[i];
n4 = lemda3[i];
n5 = lemda3[i];
n7 = lemda3[i];

b3++;
break;

case 8:
n3 = lemda3[i];
n2 = lemda3[i];
n4 = lemda3[i];
n5 = lemda3[i];
n7 = lemda3[i];
n8 = lemda3[i];

b3++;
break;

case 9:
n3 = lemda3[i];
n6 = lemda3[i];
n10 = lemda3[i];
n9 = lemda3[i];

b3++;
```

```
break;

case 10:
n6 = lemda3[i];
n3 = lemda3[i];
n10 = lemda3[i];

b3++;
break;

case 11:
n3 = lemda3[i];
n2 = lemda3[i];
n4 = lemda3[i];
n11 = lemda3[i];

b3++;
break;

case 12:
n2 = lemda3[i];
n4 = lemda3[i];
n11 = lemda3[i];
n12 = lemda3[i];
b3++;
break;

case 13:
n2 = lemda3[i];
n4 = lemda3[i];
n11 = lemda3[i];
n12 = lemda3[i];
n13 = lemda3[i];
b3++;
break;

case 14:
n2 = lemda3[i];
n4 = lemda3[i];
n5 = lemda3[i];
n7 = lemda3[i];
n8 = lemda3[i];
n9 = lemda3[i];
n14 = lemda3[i];
```

```
                        b3++;
                        break;
                        }}
int lemda4[] = {7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,1,2,3,4,5,6};

                        Random src4 = new Random();
                        int num4;
                        b4=0;
                        for (count4=1; count4<=14; count4++) {
                        num4 = 1+src4.nextInt(14);
                        switch(num4) {

                        case 1:
                              n1 = lemda4[i];
                              n2 = lemda4[i];
                              n4 = lemda4[i];
                              b4++;
                              break;

                              case 2:
                              n2 = lemda4[i];
                              n4 = lemda4[i];
                              b4++;
                              break;

                              case 3:
                              n2 = lemda4[i];
                              n3 = lemda4[i];
                              n4 = lemda4[i];
                              b4=b4+1;
                              break;

                              case 4:

                              break;

                              case 5:

                              n5 = lemda4[i];
                              n4 = lemda4[i];

                              b4=b4+1;
                              break;
```

```
case 6:
n6 = lemda4[i];
n5 = lemda4[i];
n4 = lemda4[i];
b4=b4+1;
break;

case 7:
n4 = lemda4[i];
n5 = lemda4[i];
n7 = lemda4[i];


b4++;
break;

case 8:
n4 = lemda4[i];
n5 = lemda4[i];
n7 = lemda4[i];
n8 = lemda4[i];

b4++;
break;

case 9:
n4 = lemda4[i];
n5 = lemda4[i];
n7 = lemda4[i];
n8 = lemda4[i];
n9 = lemda4[i];


b4++;
break;

case 10:
n4 = lemda4[i];
n5 = lemda4[i];
n6 = lemda4[i];
n10 = lemda4[i];
b4++;
break;
```

```
                    case 11:
                    n4 = lemda4[i];
                    n11 = lemda4[i];

                    b4++;
                    break;

                    case 12:
                    n4 = lemda4[i];
                    n11 = lemda4[i];
                    n12 = lemda4[i];
                    b4++;
                    break;

                    case 13:
                    n4 = lemda4[i];
                    n11 = lemda4[i];
                    n12 = lemda4[i];
                    n13 = lemda4[i];
                    b4++;
                    break;

                    case 14:
                    n4 = lemda4[i];
                    n5 = lemda4[i];
                    n7 = lemda4[i];
                    n8 = lemda4[i];
                    n9 = lemda4[i];
                    n14 = lemda4[i];
                    b4++;
                    break;
                    }}

int lemda5[] = {9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,1,2,3,4,5,6,7,8};

            Random src5 = new Random();
            int num5;
            b4=0;
            for (count4=1; count4<=14; count4++) {
            num5 = 1+src5.nextInt(14);
            switch(num5) {

                    case 1:
                    n1 = lemda5[i];
```

```
n2 = lemda5[i];
n4 = lemda5[i];
n5 = lemda5[i];
b5++;
break;

case 2:
n2 = lemda5[i];
n4 = lemda5[i];
n5 = lemda5[i];
b5++;
break;

case 3:
n2 = lemda5[i];
n3 = lemda5[i];
n4 = lemda5[i];
n5 = lemda5[i];
b5++;
break;

case 4:
n4 = lemda5[i];
n5 = lemda5[i];
b5++;
break;

case 5:

break;

case 6:
n6 = lemda5[i];
n5 = lemda5[i];
b5++;
break;

case 7:
n5 = lemda5[i];
n7 = lemda5[i];


b5++;
break;
```

```
case 8:
n5 = lemda5[i];
n7 = lemda5[i];
n8 = lemda5[i];

b5++;
break;

case 9:
n5 = lemda5[i];
n7 = lemda5[i];
n8 = lemda5[i];
n9 = lemda5[i];


b5++;
break;

case 10:
n5 = lemda5[i];
n6 = lemda5[i];
n10 = lemda5[i];
b5++;
break;

case 11:
n11 = lemda5[i];
n4 = lemda5[i];
n5 = lemda5[i];
b5++;
break;

case 12:
n5 = lemda5[i];
n7 = lemda5[i];
n8 = lemda5[i];
n8 = lemda5[i];
n12 = lemda5[i];
b5++;
break;

case 13:
        n5 = lemda5[i];
```

```
                        n7 = lemda5[i];
                        n8 = lemda5[i];
                        n9 = lemda5[i];
                        n12 = lemda5[i];
                        n13 = lemda5[i];

                        b5++;
                break;

                case 14:
                n5 = lemda5[i];
                n7 = lemda5[i];
                n8 = lemda5[i];
                n9 = lemda5[i];
                n14 = lemda5[i];
                b5++;
                break;
                }}

int lemda6[] = {11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,1,2,3,4,5,6,7,8,9,10};

        Random src6 = new Random();
        int num6;
        b6=0;
        for (count6=1; count6<=14; count6++) {
        num6 = 1+src4.nextInt(14);
        switch(num6) {

        case 1:
                n1 = lemda6[i];
                n2 = lemda6[i];
                n3 = lemda6[i];
                n6 = lemda6[i];
                b6++;
                break;

                case 2:
                n2 = lemda6[i];
                n3 = lemda6[i];
                n6 = lemda6[i];
                b6++;
                break;

                case 3:
```

```
n3 = lemda6[i];
n6 = lemda6[i];
b6++;
break;

case 4:
n4 = lemda6[i];
n5 = lemda6[i];
n6 = lemda6[i];
b6++;
break;
case 5:
            n5 = lemda6[i];
            n6 = lemda6[i];
            b6++;

break;

case 6:
break;

case 7:
            n6 = lemda6[i];
            n5 = lemda6[i];
n7 = lemda6[i];


b6++;
break;

case 8:
            n6 = lemda6[i];
            n5 = lemda6[i];
n7 = lemda6[i];
n8 = lemda6[i];

b6++;
break;

case 9:
n6 = lemda6[i];
n10 = lemda6[i];
n9 = lemda6[i];
```

```
                        b6++;
                        break;

                        case 10:
                        n6 = lemda6[i];
                        n10 = lemda6[i];
                        b6++;
                        break;
                case 11:
                        n11 = lemda6[i];
                        n6 = lemda6[i];
                        n10 = lemda6[i];
                        n9 = lemda6[i];
                        n12 = lemda6[i];
                        b6++;
                        break;

                        case 12:
                        n6 = lemda6[i];
                        n10 = lemda6[i];
                        n9 = lemda6[i];
                        n12 = lemda6[i];
                        b5++;
                        break;

                        case 13:
                                n6 = lemda6[i];
                                n13 = lemda6[i];

                                b6++;
                        break;

                        case 14:
                        n6 = lemda5[i];
                        n14 = lemda5[i];
                        b6++;
                        break;
                        }}
        int lemda7[] = {13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,1,2,3,4,5,6,7,8,9,10,11,12};

                Random src7 = new Random();
                int num7;
                b7=0;
```

```
for (count7=1; count7<=14; count7++) {
num7 = 1+src7.nextInt(14);
switch(num7) {

case 1:
        n1 = lemda7[i];
        n2 = lemda7[i];
        n4 = lemda7[i];
        n5 = lemda7[i];
        n7 = lemda7[i];
        b7++;
        break;

        case 2:
        n2 = lemda7[i];
        n4 = lemda7[i];
        n5 = lemda7[i];
        n7 = lemda7[i];
        b7++;
        break;

        case 3:
        n3 = lemda7[i];
        n2 = lemda7[i];
        n4 = lemda7[i];
        n5 = lemda7[i];
        n7 = lemda7[i];

        b7++;
        break;

        case 4:
        n4 = lemda7[i];
        n5 = lemda7[i];
        n7 = lemda7[i];
        b7++;
        break;

        case 5:
                n5 = lemda7[i];
                n7 = lemda7[i];
                b7++;

        break;
```

```
case 6:
            n6 = lemda7[i];
            n5 = lemda7[i];
            n7 = lemda7[i];
            b7++;

            break;

case 7:

break;

case 8:
n7 = lemda7[i];
n8 = lemda7[i];

b7++;
break;

case 9:
n7 = lemda7[i];
n8 = lemda7[i];
n9 = lemda7[i];


b7++;
break;

case 10:
            n7 = lemda7[i];
            n8 = lemda7[i];
            n9 = lemda7[i];
            n10 = lemda7[i];


            b7++;
            break;

case 11:
            n7 = lemda7[i];
            n8 = lemda7[i];
            n9 = lemda7[i];
            n12 = lemda7[i];
```

```
                    n11 = lemda7[i];
                    b7++;
                    break;

                    case 12:
                              n7 = lemda7[i];
                              n8 = lemda7[i];
                              n9 = lemda7[i];
                              n12 = lemda7[i];
                    b7++;
                    break;

                    case 13:
                              n7 = lemda7[i];
                              n8 = lemda7[i];
                              n9 = lemda7[i];
                              n12 = lemda7[i];
                              n13 = lemda7[i];

                              b7++;
                    break;

                    case 14:
                              n7 = lemda7[i];
                              n8 = lemda7[i];
                              n9 = lemda7[i];
                              n14 = lemda7[i];
          b7++;
                    break;
                    }}

int lemda8[] = {15,16,17,18,19,20,21,22,23,24,25,26,27,28,1,2,3,4,5,6,7,8,9,10,11,12,13,14};

          Random src8 = new Random();
          int num8;
          b8=0;
          for (count8=1; count8<=14; count8++) {
          num8 = 1+src8.nextInt(14);
          switch(num8) {

          case 1:
          n1 = lemda8[i];
          n8 = lemda8[i];
          b8++;
```

```
break;

case 2:
n2 = lemda8[i];
n4 = lemda8[i];
n5 = lemda8[i];
n7 = lemda8[i];
n8 = lemda8[i];

b8++;
break;

case 3:
n2 = lemda3[i];
n3 = lemda3[i];
n4 = lemda8[i];
n5 = lemda8[i];
n7 = lemda8[i];
n8 = lemda8[i];

b8++;

break;

case 4:
        n4 = lemda8[i];
        n5 = lemda8[i];
        n7 = lemda8[i];
        n8 = lemda8[i];
b8++;
break;
case 5:
        n5 = lemda8[i];
        n7 = lemda8[i];
        n8 = lemda8[i];

        b8++;

break;

case 6:
        n6 = lemda8[i];
        n5 = lemda8[i];
        n7 = lemda8[i];
        n8 = lemda8[i];

        b8++;

        break;

case 7:
        n7 = lemda8[i];
        n8 = lemda8[i];
        b8++;
break;

case 8:
break;

case 9:
n8 = lemda8[i];
n9 = lemda8[i];


b8++;
break;

case 10:
        n8 = lemda8[i];
        n9 = lemda8[i];
        n10 = lemda8[i];


        b8++;
        break;

case 11:
        n11 = lemda8[i];
        n8 = lemda8[i];
        n9 = lemda8[i];
        n12 = lemda8[i];

b8++;
break;

case 12:
        n8 = lemda8[i];
        n9 = lemda8[i];
```

```
                        n12 = lemda8[i];
                b8++;
                break;

                case 13:
                        n8 = lemda8[i];
                        n9 = lemda8[i];
                        n12 = lemda8[i];
                        n13 = lemda8[i];

                        b8++;
                break;

                case 14:
                        n8 = lemda8[i];
                        n9 = lemda8[i];
                        n14 = lemda8[i];
        b8++;
                break;
                }}


int lemda9[] = {17,18,19,20,21,22,23,24,25,26,27,28,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16};

                Random src9 = new Random();
                int num9;
                b9=0;
                for (count9=1; count9<=14; count9++) {
                num9 = 1+src9.nextInt(14);
                switch(num9) {

                case 1:
                        n1 = lemda9[i];
                        n8 = lemda9[i];
                        n9 = lemda9[i];

                        b9++;
                        break;

                        case 2:
                        n2 = lemda9[i];
                        n4 = lemda9[i];
                        n5 = lemda9[i];
                        n7 = lemda9[i];
```

```
n8 = lemda9[i];
n9 = lemda9[i];

b9++;
break;

case 3:
n3 = lemda9[i];
n6 = lemda9[i];
n10 = lemda9[i];
n7 = lemda9[i];
n9 = lemda9[i];

b9++;

break;

case 4:
        n4 = lemda9[i];
        n5 = lemda9[i];
        n7 = lemda9[i];
        n9 = lemda9[i];
        n8 = lemda9[i];
b9++;
break;
case 5:
        n5 = lemda9[i];
        n7 = lemda9[i];
        n8 = lemda9[i];
        n9 = lemda9[i];

        b9++;

break;

case 6:
        n6 = lemda9[i];
        n10 = lemda9[i];
        n9 = lemda9[i];

        b9++;

        break;
```

```
case 7:
        n7 = lemda9[i];
        n8 = lemda9[i];
        n9 = lemda9[i];

        b9++;
break;

case 8:
        n8 = lemda9[i];
        n9 = lemda9[i];
        b9++;
        break;

case 9:
break;

case 10:
        n9 = lemda9[i];
        n10 = lemda9[i];


        b9++;
        break;

case 11:
        n11 = lemda9[i];
        n8 = lemda9[i];
        n9 = lemda9[i];
        n12 = lemda9[i];

b9++;
break;

case 12:
        n9 = lemda9[i];
        n12 = lemda9[i];
b9++;
break;

case 13:
        n9 = lemda9[i];
        n14 = lemda9[i];
        n13 = lemda9[i];
```

```
                    b9++;
                break;

                case 14:
                    n9 = lemda9[i];
                    n14 = lemda9[i];
            b9++;
                break;
                }}


int lemda10[] = {19,20,21,22,23,24,25,26,27,28,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18};

            Random src10 = new Random();
            int num10;
            b10=0;
            for (count10=1; count10<=14; count10++) {
            num10 = 1+src10.nextInt(14);
            switch(num10) {

            case 1:
                n1 = lemda10[i];
                n8 = lemda10[i];
                n9 = lemda10[i];
                n10 = lemda10[i];
                b10++;
                break;

                case 2:
                n2 = lemda10[i];
                n3 = lemda10[i];
                n6 = lemda10[i];
                n10 = lemda10[i];

                b10++;
                break;

                case 3:
                n3 = lemda10[i];
                n6 = lemda10[i];
                n10 = lemda10[i];

                b10++;
```

```
break;

case 4:
            n4 = lemda10[i];
            n5 = lemda10[i];
            n6 = lemda10[i];
            n10 = lemda10[i];
            b10++;
break;
case 5:
            n5 = lemda10[i];
            n6 = lemda10[i];
            n10 = lemda10[i];

            b10++;

break;

case 6:
            n6 = lemda10[i];
            n10 = lemda10[i];

            b10++;

            break;

case 7:
            n7 = lemda10[i];
            n8 = lemda10[i];
            n9 = lemda10[i];
            n10 = lemda10[i];

            b10++;
break;

case 8:
            n8 = lemda10[i];
            n9 = lemda10[i];
            n10 = lemda10[i];

            b10++;
            break;
```

```
        case 9:
                n9 = lemda10[i];
                n10 = lemda10[i];
                b10++;

        break;


        case 10:
        break;

        case 11:
                n11 = lemda10[i];
                n8 = lemda10[i];
                n9 = lemda10[i];
                n10 = lemda10[i];

        b10++;
        break;

        case 12:
                n9 = lemda10[i];
                n10 = lemda10[i];
                n12 = lemda10[i];
        b10++;
        break;

        case 13:
                n9 = lemda10[i];
                n14 = lemda10[i];
                n13 = lemda10[i];
                n10 = lemda10[i];

                b10++;
        break;

        case 14:
                n9 = lemda10[i];
                n10 = lemda10[i];
                n14 = lemda10[i];
b10++;
        break;
        }}
```

```
int lemda11[] = {21,22,23,24,25,26,27,28,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20};

        Random src11 = new Random();
        int num11;
        b11=0;
        for (count11=1; count11<=14; count11++) {
        num11 = 1+src11.nextInt(14);
        switch(num11) {

        case 1:
                n1 = lemda11[i];
                n2 = lemda11[i];
                n4 = lemda11[i];
                n11 = lemda11[i];
                b10++;
                break;

                case 2:
                n2 = lemda11[i];
                n4 = lemda11[i];
                n11 = lemda11[i];

                b11++;
                break;

                case 3:
                n3 = lemda11[i];
                n2 = lemda11[i];
                n4 = lemda11[i];
                n11 = lemda11[i];

                b11++;

                break;

                case 4:
                        n4 = lemda11[i];
                        n11 = lemda11[i];
                        b11++;
                break;
                case 5:
                        n5 = lemda11[i];
                        n11 = lemda11[i];
                        n4 = lemda11[i];
        b11++;

break;

case 6:
        n6 = lemda11[i];
        n10 = lemda11[i];
        n9 = lemda11[i];
        n12 = lemda11[i];
        n11 = lemda11[i];

        b11++;

        break;

case 7:
        n7 = lemda11[i];
        n8 = lemda11[i];
        n9 = lemda11[i];
        n12 = lemda11[i];
        n11 = lemda11[i];

        b11++;
break;

case 8:
        n8 = lemda11[i];
        n9 = lemda11[i];
        n12 = lemda11[i];
        n11 = lemda11[i];

        b11++;
        break;

case 9:
        n9 = lemda11[i];
        n12 = lemda11[i];
        n11 = lemda11[i];

        b11++;

break;
```

```
                case 10:
                        n10 = lemda11[i];
                        n9 = lemda11[i];
                        n14 = lemda11[i];
                        n11 = lemda11[i];
                b11++;
                        break;

                case 11:
                break;

                case 12:
                        n11 = lemda11[i];
                        n12 = lemda11[i];
                b11++;
                break;

                case 13:
                        n11 = lemda11[i];
                        n12 = lemda11[i];
                        n13 = lemda11[i];

                        b11++;
                break;

                case 14:
                        n11 = lemda11[i];
                        n14 = lemda11[i];
        b11++;
                break;
                }}

int lemda12[] = {23,24,25,26,27,28,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22};

        Random src12 = new Random();
        int num12;
        b12=0;
        for (count12=1; count12<=14; count12++) {
        num12 = 1+src12.nextInt(14);
        switch(num12) {

        case 1:
                n1 = lemda12[i];
```

```
n8 = lemda12[i];
n8 = lemda12[i];
n12 = lemda12[i];
b12++;
break;

case 2:
n2 = lemda12[i];
n4 = lemda12[i];
n11 = lemda12[i];
n12 = lemda12[i];
b12++;
break;

case 3:
n3 = lemda12[i];
n6 = lemda12[i];
n10 = lemda12[i];
n9 = lemda12[i];
n12 = lemda12[i];

b12++;

break;

case 4:
        n4 = lemda12[i];
        n11 = lemda12[i];
        n12 = lemda12[i];

        b12++;
break;
case 5:
        n5 = lemda12[i];
        n7 = lemda12[i];
        n8 = lemda12[i];
        n9 = lemda12[i];
        n12 = lemda12[i];

        b12++;

break;

case 6:
        n6 = lemda12[i];
        n10 = lemda12[i];
        n9 = lemda12[i];
        n12 = lemda12[i];

        b12++;

        break;

case 7:
        n7 = lemda12[i];
        n8 = lemda12[i];
        n9 = lemda12[i];
        n12 = lemda12[i];

        b12++;
break;

case 8:
        n8 = lemda12[i];
        n9 = lemda12[i];
        n12 = lemda12[i];

        b12++;
        break;

case 9:
        n9 = lemda12[i];
        n12 = lemda12[i];

        b12++;

break;


case 10:
        n10 = lemda12[i];
        n9 = lemda12[i];
        n12 = lemda12[i];
        b11++;
        break;

case 11:
        n11 = lemda12[i];
```

```
                    n12 = lemda12[i];
                    b12++;
            break;

            case 12:

            break;

            case 13:
                    n12 = lemda12[i];
                    n13 = lemda12[i];

                    b12++;
            break;

            case 14:
                    n13 = lemda12[i];
                    n12 = lemda12[i];
                    n14 = lemda12[i];
    b12++;
            break;
            }}

int lemda13[] = {25,26,27,28,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24};

        Random src13 = new Random();
        int num13;
        b13=0;
        for (count13=1; count13<=14; count13++) {
        num13 = 1+src13.nextInt(14);
        switch(num13) {

        case 1:
            n1 = lemda13[i];
            n8 = lemda13[i];
            n9 = lemda13[i];
            n14 = lemda13[i];
            n13 = lemda13[i];
            b13++;
            break;

            case 2:
            n2 = lemda13[i];
            n4 = lemda13[i];
```

```
n11 = lemda13[i];
n12 = lemda13[i];
n13 = lemda13[i];
b13++;
break;

case 3:
n3 = lemda13[i];
n6 = lemda13[i];
n10 = lemda13[i];
n9 = lemda13[i];
n14 = lemda13[i];
n13 = lemda13[i];
b13++;

break;

case 4:
        n4 = lemda13[i];
        n11 = lemda13[i];
        n12 = lemda13[i];
        n13 = lemda13[i];
        b13++;
break;
case 5:
        n5 = lemda13[i];
        n7 = lemda13[i];
        n8 = lemda13[i];
        n9 = lemda13[i];
        n12 = lemda13[i];
        n13 = lemda13[i];
        b13++;

break;

case 6:
        n6 = lemda13[i];
        n13 = lemda13[i];
        b13++;

        break;

case 7:
        n7 = lemda13[i];
```

```
            n8 = lemda13[i];
            n9 = lemda13[i];
            n12 = lemda13[i];
            n13 = lemda13[i];
            b13++;
break;

case 8:
            n8 = lemda13[i];
            n9 = lemda13[i];
            n12 = lemda13[i];
            n13 = lemda13[i];
            b13++;
            break;

case 9:
            n9 = lemda13[i];
            n14 = lemda13[i];
            n13 = lemda13[i];
            b13++;

break;


case 10:
            n10 = lemda13[i];
            n9 = lemda13[i];
            n12 = lemda13[i];
            n13 = lemda13[i];
            b13++;
            break;

case 11:
            n11 = lemda13[i];
            n12 = lemda13[i];
            n13 = lemda13[i];
            b13++;
break;

case 12:
            n12 = lemda13[i];
            n13 = lemda13[i];
            b13++;
break;
```

```
                    case 13:

                    break;

                    case 14:
                              n13 = lemda13[i];
                              n14 = lemda13[i];

          b13++;
                    break;
                    }}

int lemda14[] = {27,28,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26};

          Random src14 = new Random();
          int num14;
          b14=0;
          for (count14=1; count14<=14; count14++) {
          num14 = 1+src14.nextInt(14);
          switch(num14) {

          case 1:
                    n1 = lemda14[i];
                    n8 = lemda14[i];
                    n9 = lemda14[i];
                    n14 = lemda14[i];
                    b14++;
                    break;

                    case 2:
                    n2 = lemda14[i];
                    n4 = lemda14[i];
                    n5 = lemda14[i];
                    n7 = lemda14[i];
                    n8 = lemda14[i];
                    n9 = lemda14[i];
                    n14 = lemda14[i];
                    b14++;
                    break;

                    case 3:
                    n3 = lemda14[i];
                    n6 = lemda14[i];
```

```
n10 = lemda14[i];
n9 = lemda14[i];
n14 = lemda14[i];
b14++;

break;

case 4:
            n4 = lemda14[i];
            n5 = lemda14[i];
            n7 = lemda14[i];
            n8 = lemda14[i];
            n9 = lemda14[i];
            n14 = lemda14[i];
            b14++;
break;
case 5:
            n5 = lemda14[i];
            n7 = lemda14[i];
            n8 = lemda14[i];
            n9 = lemda14[i];
            n12 = lemda14[i];
            n14 = lemda14[i];
            b14++;

break;

case 6:
            n6 = lemda14[i];
            n13 = lemda14[i];
            n14 = lemda14[i];
            b14++;

            break;

case 7:
            n7 = lemda14[i];
            n8 = lemda14[i];
            n9 = lemda14[i];
            n12 = lemda14[i];
            n14 = lemda14[i];
            b14++;
break;
```

```
        case 8:
                n8 = lemda14[i];
                n9 = lemda14[i];
                n14 = lemda14[i];
                b14++;
                break;

        case 9:
                n9 = lemda14[i];
                n14 = lemda14[i];

                b14++;

        break;


        case 10:
                n10 = lemda14[i];
                n9 = lemda14[i];
                n14 = lemda14[i];
                b14++;
                break;

        case 11:
                n11 = lemda14[i];
                n14 = lemda14[i];
                b14++;
        break;

        case 12:
                n12 = lemda14[i];
                n13 = lemda14[i];
                n14 = lemda14[i];
                b14++;
        break;

        case 13:
                n13 = lemda14[i];
                n14 = lemda14[i];

b14++;
        break;

        case 14:
```

```
                                break;
                                }}
// condition is checked for wavelength contention
if (lemda1[i]==lemda2[i] || lemda1[i]==lemda3[i] ||lemda1[i]==lemda4[i] || lemda1[i]==lemda5[i] ||
lemda1[i]==lemda6[i] || lemda1[i]==lemda7[i] || lemda1[i]==lemda8[i] || lemda1[i]==lemda9[i] ||
lemda1[i]==lemda10[i] || lemda1[i]==lemda11[i] || lemda1[i]==lemda12[i] || lemda1[i]==lemda13[i] ||
lemda1[i]==lemda14[i] || lemda2[i]==lemda3[i] || lemda2[i]==lemda4[i] || lemda2[i]==lemda5[i] ||
lemda2[i]==lemda6[i] || lemda2[i]==lemda7[i] || lemda2[i]==lemda8[i] || lemda2[i]==lemda9[i] ||
lemda2[i]==lemda10[i] || lemda2[i]==lemda11[i] || lemda2[i]==lemda12[i] || lemda2[i]==lemda13[i] ||
lemda2[i]==lemda14[i] || lemda3[i]==lemda4[i] ||  lemda3[i]==lemda5[i] ||  lemda3[i]==lemda6[i] ||
lemda3[i]==lemda7[i] || lemda3[i]==lemda8[i] ||  lemda3[i]==lemda9[i] || lemda3[i]==lemda10[i] ||
lemda3[i]==lemda11[i] ||  lemda3[i]==lemda12[i] ||  lemda3[i]==lemda13[i] ||  lemda3[i]==lemda14[i] ||
lemda4[i]==lemda5[i] || lemda4[i]==lemda6[i] || lemda4[i]==lemda7[i] || lemda4[i]==lemda8[i] ||
lemda4[i]==lemda9[i] ||  lemda4[i]==lemda10[i] || lemda4[i]==lemda11[i] || lemda4[i]==lemda12[i] ||
lemda4[i]==lemda13[i] || lemda4[i]==lemda14[i] || lemda5[i]==lemda6[i] || lemda5[i]==lemda7[i] ||
lemda5[i]==lemda8[i] || lemda5[i]==lemda9[i] || lemda5[i]==lemda10[i] || lemda5[i]==lemda11[i] ||
lemda5[i]==lemda12[i] || lemda5[i]==lemda13[i] || lemda5[i]==lemda14[i] || lemda6[i]==lemda7[i] ||
lemda6[i]==lemda8[i] || lemda6[i]==lemda9[i] || lemda6[i]==lemda10[i] || lemda6[i]==lemda11[i] ||
lemda6[i]==lemda12[i] || lemda6[i]==lemda13[i] || lemda6[i]==lemda14[i] || lemda7[i]==lemda8[i] ||
lemda7[i]==lemda9[i] || lemda7[i]==lemda10[i] || lemda7[i]==lemda11[i] || lemda7[i]==lemda12[i] ||
lemda7[i]==lemda13[i] || lemda7[i]==lemda14[i] || lemda8[i]==lemda9[i] || lemda8[i]==lemda10[i] ||
lemda8[i]==lemda11[i] || lemda8[i]==lemda12[i] || lemda8[i]==lemda13[i] || lemda8[i]==lemda14[i] ||
lemda9[i]==lemda10[i] || lemda9[i]==lemda11[i] || lemda9[i]==lemda12[i] || lemda9[i]==lemda13[i] ||
lemda9[i]==lemda14[i] || lemda11[i]==lemda12[i] ||lemda11[i]==lemda13[i] ||lemda11[i]==lemda14[i]
||lemda12[i]==lemda13[i] ||lemda13[i]==lemda14[i])
        bl++;}
System.out.println("burst loss "+bl);
System.out.println("burst counter1 "+b1);
System.out.println("burst counter2 "+b2);
System.out.println("burst counter3 "+b3);
System.out.println("burst counter4 "+b4);
System.out.println("burst counter5 "+b5);
System.out.println("burst counter6 "+b6);
System.out.println("burst counter7 "+b7);
System.out.println("burst counter8 "+b8);
System.out.println("burst counter9 "+b9);
System.out.println("burst counter10 "+b10);
System.out.println("burst counter11 "+b11);
System.out.println("burst counter12 "+b12);
System.out.println("burst counter13 "+b13);
System.out.println("burst counter14 "+b14);

}}}
```