NTNU

Norwegian University of
Science and Technology

OTN switching

Per Harald Knudsen-Baas

Master of Telematics - Communication Networks and Networked Services (2 year)

Submission date: June 2011

Supervisor: Steinar Bjørnstad, ITEM

Co-supervisor: Raimena Veisllari, ITEM

Norwegian University of Science and Technology
Department of Telematics

Problem description

The OTN (Optical Transport Network) standard is an ITU-T standard (G.709) describing a method for wrapping in signals of different protocol-formats for transport across an optical network. OTN is considered as the predecessor of SDH, enabling much of the same monitoring and management capabilities known from SDH. While the first versions of the OTN-standard describes transport at 2.5 and 10 Gb/s wavelength-channel bitrates, the standard has recently been extended to include bitrates up to 100 Gb/s and down to 1 Gb/s. Furthermore, while OTN originally were only described as a method for reliable data-transport, switching of sub-wavelength bitrates has recently been proposed. This enables e.g. add/drop at the OTN layer of parts of e.g. a 100 Gb/s stream, enabling the use of 10 Gb/s interfaces on IP-routers in a 100 Gb/s transport network.

The thesis will study recent progress in OTN, including OTN switching capabilities. Performance of a pure packet switched network shall be compared with the performance of an OTN switching based network. The performance comparison will be performed on a proposed network scenario. Performance results shall be found using discrete event simulation.

Assignment given: 17.01.2011
Supervisor: Steinar Bjørnstad, ITEM/Transpacket

Abstract

Increasing traffic volumes in the Internet put strict requirements to the architecture of optical core networks. The exploding number of Internet users, and massive increase in Internet content consumption forces carriers to constantly upgrade and transform their core networks in order to cope with the traffic growth. The choice of both physical components and transport protocols in the core network is crucial in order to provide satisfactorily performance.

Data traffic in the core network consists of a wide variety of protocols. OTN is a digital wrapper technology, responsible for encapsulating existing frames of data, regardless of native protocol, and adding additional overhead for addressing, OAM and error control. The wrapped signal is then transported directly over wavelengths in the optical transport network. The common OTN wrapper overhead makes it possible to monitor and control the signals, regardless of the protocol type being transported.

OTN is standardized by the ITU through a series of recommendations, the two most important being ITU-T G.709 - "Interfaces for the Optical Transport Network", and ITU-T G.872 - "Architecture of the Optical Transport Network". OTN uses a flexible TDM hierarchy in order to provide high wavelength utilization. The TDM hierarchy makes it possible to perform switching at various sub-wavelength bit rates in network nodes.

An introduction to OTN and an overview of recent progress in OTN standardization is given in the thesis. An OTN switch which utilizes the flexible multiplexing hierarchy of OTN is proposed, and its characteristics is tested in a network scenario, comparing it to the packet switched alternative.

Simulation results reveal that OTN switching doesn't provide any performance benefits compared to packet switching in the core network. OTN switches do however provide bypass of intermediate IP routers, reducing the requirements for router processing power in each network node. This reduces overall cost, and improves network scalability.

An automatically reconfigurable OTN switch which rearranges link subcapacities based on differences in output buffer queue lengths is also proposed and simulated in the thesis. Simulation results show that the reconfigurable OTN switch has better performance than both pure packet switching and regular OTN switching in the network scenario.

Preface

I want to thank my supervisor Steinar Bjørnstad for his valuable support during the work with this thesis. His guidance has helped me keep on the right path from start to end. I also want to thank my co-supervisor Raimena Veisllari for helpfull hints and support during the spring.

Contents

List of Figures

List of Tables

Abbreviations

AMP	Asynchronous Mapping Procedure
ATM	Asynchronous Transfer Mode
BER	Bit Error Rate
BMP	Bit-synchronous Mapping Procedure
CAIDA	Cooperative Association for Internet Data Analysis
CBR	Constant Bit Rate
DWDM	Dense Wavelength Division Multiplexing
FAS	Frame Alignment Signal
FEC	Forward Error Correction
GFP	Generic Framing Procedure
IaDI	Intra-domain Interface
IEEE	Institute of Electrical and Electronics Engineers
IrDI	Inter-domain Interface
ITU	International Telecommunication Union
MPLS	Multiprotocol Label Switching
OCC	Optical Channel Carrier
OCG	Optical Channel Group
OCh	Optical Channel
ODTU	Optical Data Tributary Unit

ODTUG Optical Data Tributary Unit Group

ODU Optical Data Unit

OMS Optical Multiplex Section

OPU Optical Packet Unit

OTH Optical Transport Hierarchy

OTN Optical Transport Network

OTS Optical Transmission Section

OTU Optical Transmission Unit

OXC Optical crossconnect

PDH Plesiochronous Digital Hierarchy

SDH Synchronous Digital Hierarchy

SNR Signal-to-Noise Ratio

SONET Synchronous Optical Networking

TCM Tandem Connection Monitoring

TDM Time Division Multiplexing

TS Tributary Slot

TSOH Tributary Slot Overhead

WDM Wavelength Division Multiplexing

Chapter 1

Introduction

1.1 Optical transport network evolution

Over the last two decades we have been witnessing a tremendous growth in generated data traffic volumes. The amount of global IP traffic is expected to quadruple from 2009 to 2014 [8]. Bandwidth-demanding applications and services are launched at an incredible pace, and customers are constantly requiring higher connection speeds.

Telecommunication networks look very different nowadays than what they did in the end of the 19-th century when the phone was invented by Mr. Bell. In the early days of telecommunication, transport networks were constructed to provide analog voice circuits with regional connectivity. The infrastructure was based on copper cables and switching was carried out by switchingboard-ladies who manually patched circuits correctly to provide connection between endpoints. Automation in the transport networks didn't happen until the digital era in the 1960's. At this time, the transport network was responsible for transporting voice signals [9]. Since then, the Internet bubble has made the core networks undergo a shift from transporting voice to transporting data. Todays core networks transport mostly data traffic consisting of a wide variety of protocols [9].

Optical networks make it possible for carriers to handle the explosive growth in traffic volumes. The enormous capacity avilable in optical fiber cables enables transport at constantly higher bit rates. 100 Gb/s per wavelength is the next step in the fiber transmission capacity evolution.

The OTN standard is developed by the ITU and is considered as the predecessor of SDH. OTN is designed with future bandwidth and protocol-demands in mind, while maintaining the advantages of SDH like flexibility and resiliency. OTN provides the possibility for sub-lambda switching, and

offers high wavelength utilization through its flexible TDM hierarchy. It is gradually replacing SDH as the new transport standard, enabling future-proof multi-wavelength transport and management capabilities.

OTN was initally only used on point-to-point links because of its strong FEC-feature, but is now used as an entirely new network layer [9]. It is used to build transparent, scalable and cost-effective networks where existing standards like Ethernet and SDH are the client signals.

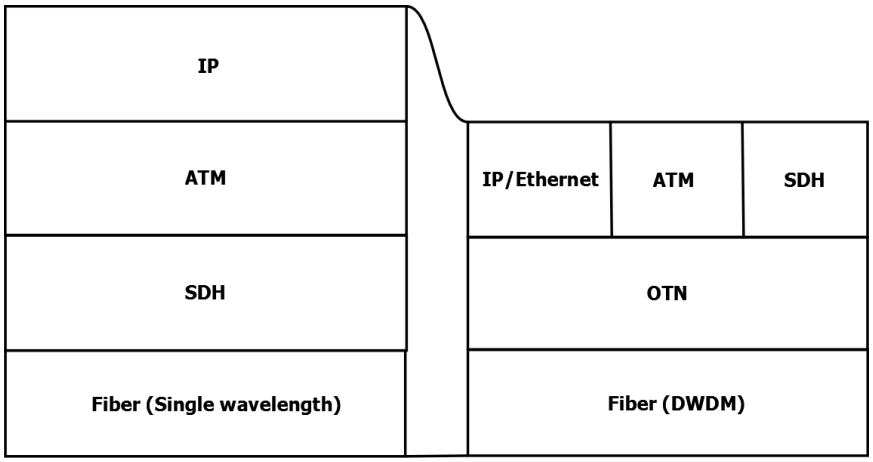

Figure 1.1: Evolution of transport network protocols; OTN provides transport for a wide range of protocols over DWDM networks.

As seen from Figure 1.1; OTN is a new core transport layer, and replaces SDH. SDH only supports client transport over a single wavelength, and isn't suitable for management of wavelength services, and addressing typical impairments in multi-wavelength optical systems. OTN accepts a wide variety of client signals (for instance Ethernet, ATM and SDH) and provides transport and management directly over DWDM networks.

Reducing the number of layers in the protocol stack has a lot of advantages. Each layer requires separate equipment, maintainance and management systems. Unnecessary duplication of functionality in the network is avoided if some layers are omitted, and cost and complexity is improved.

As the interconnection bandwidth between backbone routers reaches tens or hundreds of Gb/s, it becomes a major goal to transport as much traffic as possible over one fiber. One way is to decrease the frequency span between wavelengths in the fiber, thus increasing the total number of wavelengths. The other approach is to increase the utilization of each wavelength by using efficient TDM techniques. OTN utilizes the capacity of each wavelength by

multiplexing lower rate signals into higher rate signals. By utilizing the TDM hierarchy in OTN, it is possible to perform sub-lambda granularity switching. This might be an attractive and future-proof way of performing switching in optical networks.

1.2 Motivation

A society is totally dependt upon basic necessities as roads and infrastructure for power transmission and water to work. A power failure might put whole communities out of action. Businesses stop working, schools and universities must temporarily shut down. Failures in the telecommunication core network have a similar impact on the society. We are living our lives in a digital, interconnected era where people rely on telecommunication systems on a daily basis. Businesses, government and individuals are dependent upon the core telecommunication infrastructure to function. Either being a phone call taking place over the mobile network, surfing the web or sending a SMS; all these services rely on the core network. Design and implementation concerning the core network is thus just as important as planning and constructing roads, power- and water supply. A modern society simply stops working if the core network shuts down.

Humans have a desire to interact. We are experiencing a rapid growth of video and content consumption while users constantly expect better services across any device, at any network, at any time. With soaring IP traffic volumes, several considerations must be made concerning the architecture of core networks. A major goal is to transport traffic as efficiently as possible from network ingress to network egress, consuming as little resources as possible. Choosing an adequate core network architecture is crucial both in terms of user percieved performance and carriers balance between investment/operational cost and profit.

There are at the same time other perspectives of technology choices. One is for instance power consumption, which involves both cost and enviromental issues. Core IP routers are known to use much power for each bit that is processed [10]. The telecommunication equipment vendor Alcatel-Lucent often uses the slogan: "Switch where you can, route where you must". Switching aggregated coarse chunks of data instead of performing routing on a per-packet basis might be the most future proof solution in core networks. This might save carriers of cost both in terms of investment (switches that are cheaper than routers) and operation (power consumption).

The ITU is an international telecom standardization association and is based in Geneva, Switzerland. The association coordinates and standardizes

global use of telecommunication facilities including e.g. satellite communication, radio networks, broadcasting and optical core networks [11].

The ITU-T is a division of the ITU consisting of different groups which is responsible for studying and creating recommendations for the telecommuication field. ITU-T standardizes the OTN through several recommendations.

With the ITU-T standards as basis, OTN switching is proposed and evaluated in this thesis. OTN sub-lambda switching might be the future solution for optimum transport efficiency in optical core networks.

1.3 Previous work

This master thesis is a continuation of the project assignment with similar title performed in the specialization project (TTM4511) at NTNU fall 2010 [12]. An OTN switch which performs switching at ODU level using a space-division TDM switch was proposed in the project assignment. Possible improvements in resource usage was evaluated using a six-node network scenario, comparing it to a packet switched scenario consisting of IP-routers with and without optical crossconnects. The evaluation clearly showed benefits in favour of OTN switching. Wavelength utilization is very high due to the flexible multiplexing architecture of OTN, thus improving the total transmission capacity. Bypass of intermediate IP-routers saves at the same time unneccessary processing of transit traffic. The traffic matrix used for evaluation purposes in the network scenario was however static, and might thus have been optimal for the use of OTN switches.

1.4 Problem definition

OTN is considered as the predecessor of SDH. It takes single wavelength SDH technology a step further, enabling transport of a wide variety of services over DWDM networks. OTN combines backward compatibility for existing protocols with the bandwidth expandibility of future transport networks, while keeping the benefits of SDH like reliability and manageability. There are currently several standardization documents defining OTN. The most important of them are ITU-T G.709 - "Interfaces for the Optical Transport Network" and ITU-T G.872 - "Architecture of the Optical Transport Network".

Switching technique used in the core network gets increasingly important as the traffic volume increases. OTN switching is based on the ITU-T OTN standards, and is a new switching method which is currently being considered as a switching alternative for optical networks. This thesis aims to give an

introduction to OTN and OTN switching. Potential benefits of utilizing OTN switching in optical core networks will be investigated and compared to traditional packet switching.

1.5 Methodology

A common network scenario will be used in order to compare OTN switching with alternative network implementations. The network scenario will be realized using two different network options; packet switching and OTN switching. A simulator in DEMOS (Discrete Event Modelling on Simula) will be developed for each of the network options. Simulation runs with the simulators will then be carried out with increasing traffic load and various traffic arrival processes. Performance of the network options will be compared based on results obtained from the simulation runs. OTN switching characteristics will be evaluated based on these results.

1.6 Organization of the report

The thesis is organized as follows; chapter 1, 2 and 3 contains background material covering optical core networks and the OTN standard. Chapter 4 proposes a possible high-level OTN switch implementation. Chapter 5 introduces a network scenario which is realized by using packet switching and OTN switching. Chapter 6 presents the DEMOS simulator implementation (simulation models) of the network scenarios. Chapter 7 presents and discusses simulation results, while the conclusion is found chapter 8.

Chapter 2

Optical core networks

Optical core networks form the backbone of telecommunication systems. This chapter gives an introduction to the most important aspects of optical core networks: WDM, TDM, basic optical components and transport protocols.

2.1 Background

Optical core networks form the basis for all forms of digital communication. Although the infrastructure is invisible for the users, it is responsible for transporting all types of traffic. Either it is voice or data originating from the mobile network, a conversation taking place over the public circuit switched telephone network, or Internet traffic. All these types of communication rely on the underlying optical core network infrastructure. It is thus not only crucial that the core network works perfectly, it also has to be available at all time. Mechanisms must ensure that the correct actions are performed if failures occur, to ensure contionous connectivity.

Optical fiber has a range of benefits that makes it the number one choice for transmission medium in core networks. First of all is the incredibly wide frequency spectrum that can be utilized for communication purposes. Very low attenuation in certain frequency areas makes it possible to send signals over very long distances (hundreds of kilometers) without the need for reamplification of the signals. The fiber offers very low bit error rates (less than 10E-11), and is immune to electromagnetic interference. Because the fiber is made of sand, it is also cheap to produce and environmentally friendly [1].

2.2 WDM transmission system

WDM is an efficient technique for utilizing the extremely wide frequency spectrum available in optical fibers. The frequency spectrum that is usefull for transmitting signals over large distances is located in the 1380 and 1550 nm region. These are the low-attenuation regions of the optical fiber with very low loss, approximately 0.2 dB/km. The usable frequency spectrum for communication purposes in the fiber sums up to a vast total of 50 Thz [1]. By using lasers that transmit at different wavelengths, and multiplexing the signals together, the available fiber frequency spectrum is utilized.

Figure 2.1 shows a block diagram of a WDM transmission system. The transmitter consists of one or more lasers which either is fixed to a certain wavelength or tunable over a range of wavelengths. Each laser is either directly modulated (the laser itself is switched on and off), or has an external modulator in order to impose data on the light stream (present 1's and 0's). An optical filter is usually used for tuning purposes. If multiple lasers are used to transmit simultaneously at different wavelengths, a multiplexer or coupler is used to combine the signals. The receiver may either consist of a tunable filter followed by a photodetector receiver, or a demultiplexer followed by several photodetectors.

The signal quality is gradually degraded when propagating through the fiber. As seen from Figure 2.1; amplifiers are used to regenerate the signals at certain intervals. There are two main categories of amplifiers; optical and electrical. Optical amplifiers keep the signal in the optical domain, and performs reamplification of the signals without any reshaping or retiming. This process is known as 1R (Regeneration). Electrical amplifiers, on the other hand, convert the signal into the electrical domain, amplifies, reshapes and retimes the signal. This process is known as 3R (Regeneration, Reshaping and Retiming).

Figure 2.1: Block diagram of a WDM transmission system [1]

Figure 2.2 exemplifies wavelength division multiplexing. Lasers tuned to different wavelengths ("colours") transmit simultaneously through the same fiber. Employing DWDM in the transmission system enables each fiber to carry up to 160 channels, depending on the channel spacing [13]. If each channel carries a 10 Gb/s signal, and 128 channels are used, the total transmission capacity of the fiber is 1,28 Tb/s. This is enough to carry up to 20 000 000 phone calls simultaneously (given the bit rate of one phone circuit; 8Khz sampling rate * 8 bits quantization = 64 Kb/s).

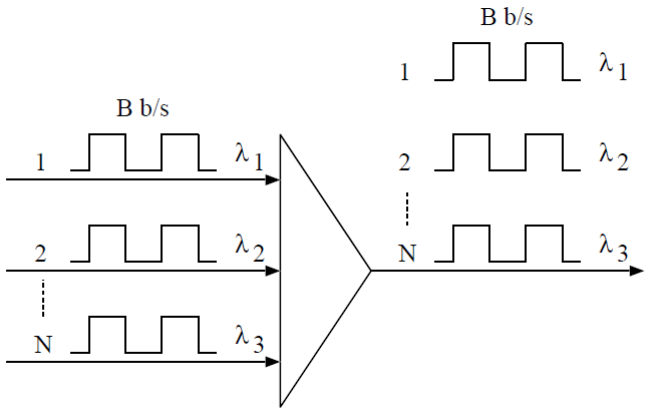

Figure 2.2: WDM: N incoming signals on separate wavelengths are multiplexed together, utilizing the available frequency spectrum in the fiber.

2.3 TDM

TDM can be used to provide higher utilization of the transmission medium. This is done by multiplexing lower-rate streams into higher-rate streams. TDM works by dividing access to the transmission medium into timeslots, and assigning incoming signals into a given timeslot in a repeating cycle. Figure 2.3 shows how TDM is used to provide higher utilization of the available transmission capacity by combining the signals coming from three different wavelengths.

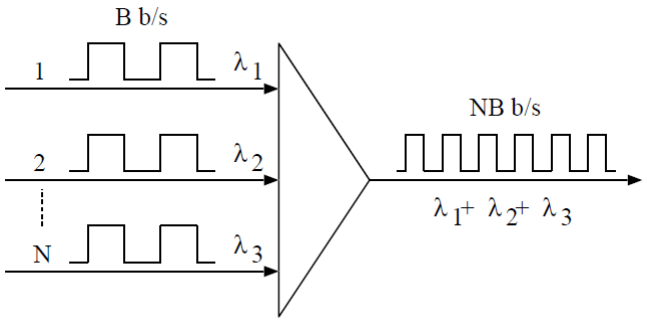

Figure 2.3: TDM: N individual streams with a bit rate of B bps on separate wavelengths are combined into a signal with bit rate NB bps. The outgoing signal is divided into N timeslots. Each of the incoming signals are assigned to a given timeslot of the outgoing signal in a repeating cycle.

2.4 Evolution of optical transmission systems

As seen from the top of Figure 2.4; early generation optical transmission systems were designed using only a single wavelength in each fiber. The bit rate was increased by using TDM on a single wavelength (explained in Figure 2.3). Electrical regenerators were used to regenerate the signal at fixed points. The regenerators first converted the signal from the optical to the electrical domain, regenerated it, and converted it back to optical. Using only a single wavelength in each connection doesn't utilize the extremely wide frequency range available in the fiber.

Current optical transmission systems is seen in the bottom part of Figure 2.4. WDM is used to utilize the available frequency spectrum in the fiber, and optical amplifiers is used instead of electrical amplifiers. In this way, the total transmission capacity is increased while cost is severly reduced.

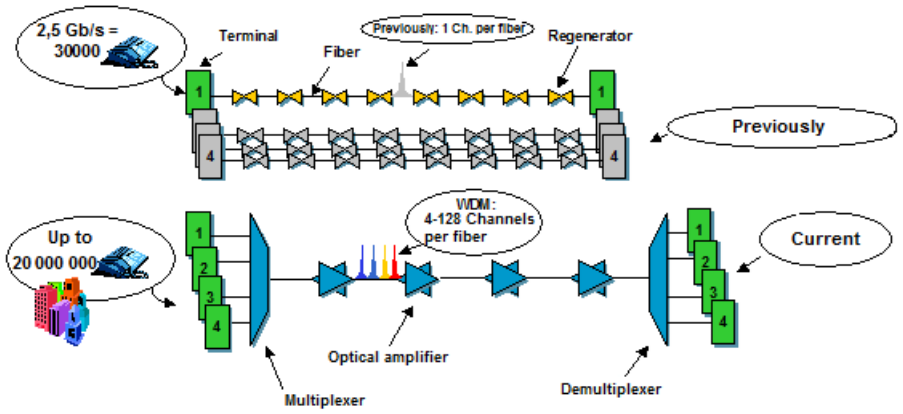

Figure 2.4: Evolution of optical transmission systems: From single wavelengths using TDM and regenerators to multiple wavelengths with WDM and optical amplifiers [2].

2.5 Crossconnects

Crossconnects are important components used to create connectivity in the optical core network. The crossconnect works by crossconnecting (switching) a single wavelength on one input to a different output. As seen from Figure 2.5; the crossconnect can either be optical or electrical[3]:

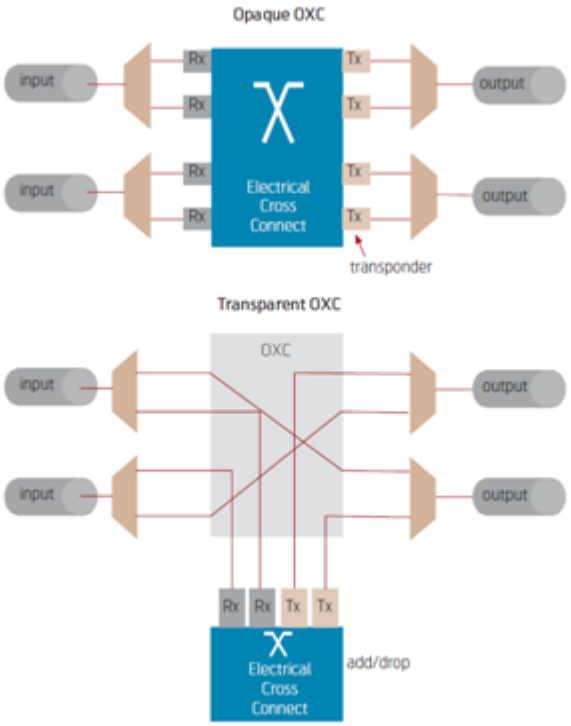

Figure 2.5: The figure shows an electrical crossconnect (top) and an optical crossconnect (bottom) [3].

Optical crossconnects are all-optical devices, and keeps the signal in the optical domain. They are transparent to bit rate and client signal and big amounts of ports are offered at a reasonable cost. It is however not possible to perform signal regeneration or monitoring when the crossconnecting is performed in the optical domain.

Electrical crossconnects converts the incoming optical wavelength to the electrical domain, cross connects it to an output port, and reconverts it back to the optical domain. The signals are regenerated when using this approach, leaving the node free of dispersion and attenuation. Digital signal monitoring is also possible to achieve. The main drawback is that the electrical crossconnect is opaque to bit rate and client signal format. Only signals with the properties supported by the crossconnect are accepted.

2.6 Core network transport protocols

Core networks span huge distances and serve thousands of people. There are thus strict requirements to performance, uptime and failure recovery capabilities in the core network. This is seen when comparing protocols used in local area networks with protocols used in core networks. Local area networks serve a limited number of users in a limited area, and thus have less stringent requirements for uptime and failure recovery. Protocols used in local area networks are thus "lightweight" compared to the transport protocols used in the core networks.

Transport protocols work as containers, and are responsible for transmitting aggregated volumes of traffic, consisting of a wide range of protocols between nodes in the core network. Transport protocols must have mechanisms for failure detection, notification, localization and recovery.

Early generation transport networks were built to carry voice circuits. The first digital transport networks were PDH networks. PDH is based on how traditional voice circuits were multiplexed together. A single voice circuit is digitized by sampling the voice signal at 8 kHz and using 8 bits quantitization. A single voice circuit thus has a bit rate of: 8 kHz * 8 bits = 64 kb/s. 64 kb/s is thus used as the basic multiplexing unit in PDH. The voice circuits were then multiplexed into aggregate signals of 2.048 Mb/s (30 voice channels, E1), 8.488 Mb/s (120 voice channels, E2), 34.368 Mb/s (480 voice channels, E3), and 139.264 Mb/s (1920 voice channels, E4) [9]. PDH had several shortcomings, and has gradually been replaced by SDH. The main problem was the asynchronous nature of PDH. Each node had its local clock, something which led to bit rate variations in the nodes. Bit stuffing had to be used in order to compensate for the bit rate variations. Bit rates in the PDH multiplexing hierarchy werent't exact multiples of 64 kb/s because of the bit stuffing. It was of this reason not possible to extract a low rate stream directly from the higher order multiplex. Instead, the whole higher order multiplex had to be demultiplexed before the single stream could be extracted. This led to the need for "multiplexer mountains" which is an inefficient, complex and costly solution [14].

Todays core networks are mainly based on SDH and OTN as client transport technology [9]. SDH solved many of the problems that PDH suffered from. All node clocks are synchronized with a common master clock in SDH. The information streams are thus exactly integer multiples of 64 kb/s. It is of this reason possible to extract low bit rate streams from higher bit rate streams without demultiplexing the whole signal. SDH also offered improved operation and maintenance procedures that PDH was missing [9].

OTN is, in contrast to SDH, asynchronous. OTN carries synchronization

data in the payload, and there is of this reason no need for a common master clock that synchronizes all the nodes in the network [5]. This simplifies the network design, and reduces cost. OTN is the next step in the evolution of transport protocols, and offers many other important and attractive features for future networking that SDH lacks. The details of OTN is given in the next chapter.

Chapter 3

OTN (G.709)

3.1 Background

As previously explained; DWDM allows for maximum utilization of the fiber transmission capacity by splicing the available frequency spectrum in the fiber into a range of wavelengths. The wavelengths serves as channels through the fiber, making it possible to transmit multiple signals, each on its own wavelength, simultaneously through the fiber. In this way, the total bandwidth is increased by simultaneous transmission of signals on several wavelengths, rather than increasing the bit rate on one of the wavelengths.

WDM technology was at an early stage when SDH was developed in the 80's. SDH was thus developed with support for transport and management of only a single wavelength. Huge advances in fiber technology was made during the 80's and 90's, including the development of DWDM[15]. Since the late 90's, there has also been a tremendous increase in Internet traffic volume [8]. Apparently, there is need for a transport standard that takes single wavelength SDH technology a step further. OTN is similar to SDH, but is designed with future protocol and bandwidth needs in mind. It offers the same reliability and manageability as SDH, but with a number of improvements. OTN makes the foundation for transparent, multi-wavelength manageable networks.

OTN is the ITU's answer for maximum utilization, manageability and wide client signal support for future DWDM networks. Many standardization documents belong in the OTN category. The two most important are the G.872 and G.709. G.872 is called "Architecture of optical transport networks". It describes the network architecture and transport technology for the OTN called the Optical Transport Hierarchy. The OTH consists of OMS, OTS and Och. G.709 is called "Interfaces of the OTN". It defines the stan-

dard interfaces and rates for high-bandwidth optical signals, and focuses on structures, interfaces and mappings. Frame format, supported client signals, multiplexing structure, and supported signal rates are found in the G.709 standard.

OTN provides a flexible multiplexing hierarchy, transparent transport of client signals with backward compatibility for existing protocols, forward error correction for increased fiber span lengths and link monitoring. An important feature is the offered switching scalability made possible by the TDM multiplexing hierarchy. The proposed OTN switch, which is presented in chapter 4, demonstrates how sub-lambda ODU switching may be used to add and/or drop various lower rate signals from the higher rate OTN multiplex.

OTN provides maximum utilization of fiber capacity by combining TDM and DWDM. TDM utilizes the capacity of a single wavelength by multiplexing low rate streams into higher rate streams, while DWDM utilizes the wide frequency spectrum in the fiber.

As previously mentioned; a difference from SDH is the fact that OTN is asynchronous. This is achieved by transporting network synchronization within the payload in the OTN frame, mainly by SDH tributaries. An OTN network element thus doesn't require synchronization interfaces or complex clocks [5]. This helps reducing both cost and complexity when designing the network.

A drawback for operators that are upgrading their infrastructure from PDH and/or SDH to OTN is that new hardware and management systems are required [5].

3.2 OTN frame structure

OTN incorporates a flexible multiplexing and mapping hierarchy in order to support a wide variety of client signals and bit rates. The multiplexing structure works as containers, accepting a wide variety of data payloads at different bit rates. Multiplexing low bit rate client signals into high bit rate signals allows for the creation of bigger data containers that are transported over a wavelength.

3.2.1 OPU-k

The OPU accepts incoming client signals of various types. Supported signals include CBR clients (such as a constant bit rate ATM cell stream, SONET or SDH signals) and packet-based clients (such as IP and MPLS packets,

and Ethernet frames)[4]. The OPU is responsible for mapping of the client signals and adding the first layer of overhead. The OPU header consists of the Payload Structure Identifier which includes the payload type and overhead bits associated with the mapping of client signals into the payload.

3.2.2 ODU-k

Overhead is added to the OPU, forming the ODU. The ODU overhead includes information for maintenance and operational functions to support optical channels. Fields found in the ODU header are: PM - path performance monitoring, FTFL - fault type and fault location, GCC - generic communications channel, APS/GCC - automatic protection switching and protection communications channel, TCM - tandem connection monitoring and a set of reserved bytes for experimental purposes.
The ODU has several bit mapping schemes to accomodate new clients in the ODU [16]. Available bit mapping schemes include AMP, BMP and GFP. AMP or BMP are used to map CBR clients into the ODU. GFP-F is used for packet-/frame-based clients [4]. The details of AMP, BMP and GFP are found in [4].

An important feature in the ODU overhead is the TCM fields, allowing up to six different monitor levels across operator domains.

3.2.3 OTU-k

The OTU consists of an ODU with additional overhead. The OTU is the lowest layer belonging to the electrical domain. It is reponsible for making the data ready for transport over an optical channel. Fields found in the OTU header are: SM - section monitoring, GCC0 - general communication channel 0 and two bytes that are reserved for future international standardization [4]. FEC is added at the end of the OTU frame. FEC is a very important feature for long-haul optical transport networks. It allows for longer signal spans without signal regeneration. This is a essential feature in e.g. sub-sea installations where long fiber spans are used. The use of active regenerators in need of power supply is costly and unpractical in these cases, and can be avoided with the use FEC.

Figure 3.1 illustrates the parts that make up the OTN frame structure.

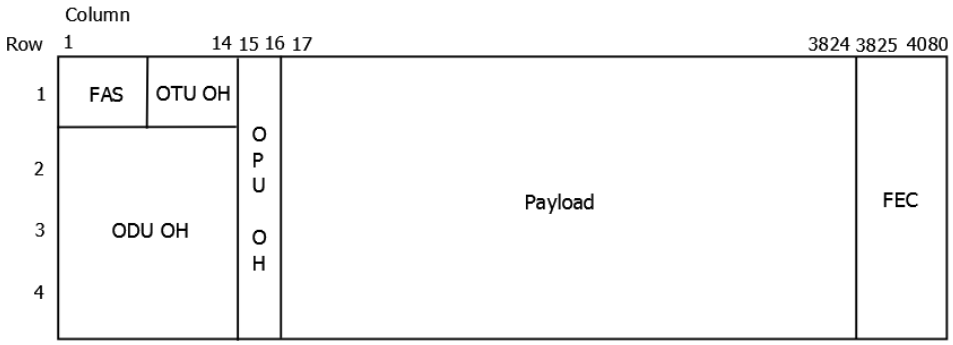

Figure 3.1: OTN frame structure showing FAS (Frame Alignment Signal), OTU overhead, ODU overhead, OPU overhead, payload and FEC

As seen from the figure; the Frame Alignment Signal is located in the top left of the frame (row 1, column 1-7), followed by the OTU overhead (row 1, column 8-14). The Frame Alignment Signal consists of six bytes, and is used to provide framing for the entire signal. A Multiframe Alignment Signal is located in in the last byte of the FAS (row 1, column 7), and is used to extend command and management functions over several frames. The MFAS counts from 0 to 255, providing a 256 multiframe structure. The ODU overhead is located in row 2-4, columns 1-14. The OPU overhead is located in columns 15-16, rows 1-4. The rest of the OTN frame consists of data payload (columns 17-3824) and FEC (columns 3825-4080).

3.2.4 OCh

The OCh is described in the ITU-T G.872 standard. It represents a wavelength in the fiber, and transports client signals between 3R regeneration points. The OCh has associated overhead in order to support management of multiple wavelengths in the OTN.

Figure 3.2 illustrates the relationship between the different OTN building blocks. As seen from the figure; overhead is added to the incoming client signal, forming the OPU. Additional overhead is then added to the OPU, forming the ODU. One to six TCM levels are added at the ODU level. Additional overhead and FEC is then added to the ODU, forming the OTU. The OTU is then transmitted over a wavelength (OCh). The OCh payload is modulated onto the OCC payload, and the OCh overhead is modulated onto the OCC overhead. Several OCC payloads can be mapped into the OMS payload and further into the OTS payload. The OCh, OMS and OTS

payloads all have their own overhead for management purposes at the optical level.

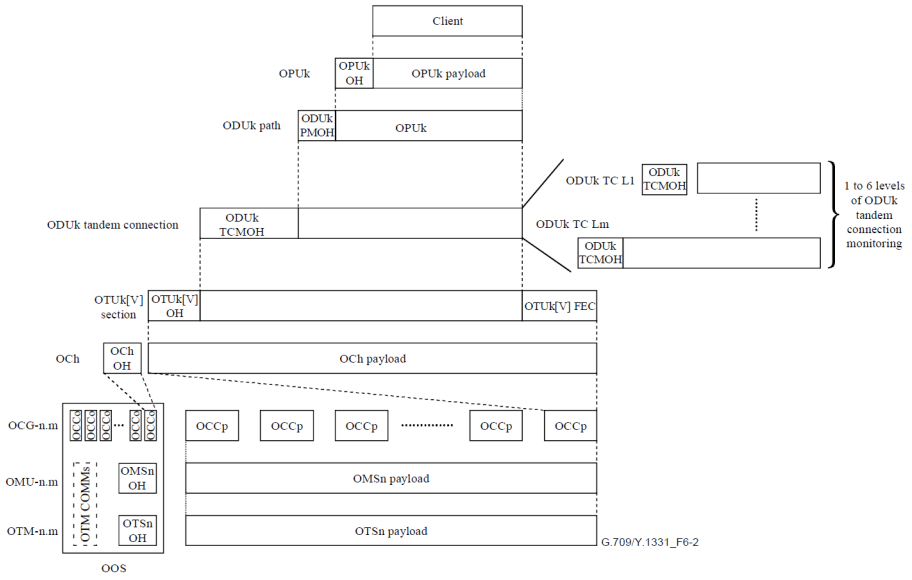

Figure 3.2: OTN structure and relationships [4]

3.2.5 ODUflex

It's hard to predict future client signal bit rates. In addition to Ethernet, many other client signals must be supported by OTN (such as Fibre Channel and video distribution signals). Most of these would not fit into any existing ODUk without significant loss of bandwidth. Defining a new ODU container for each new client signal that should be supported was considered to be unpractical [17].

To ensure flexible and easy adaptation for support of future client-signal rates, an extension known as ODUflex has been introduced to the OTN standard. ODUflex is a flexible lower order container that can be right sized to fit any client rate. It does this by occupying a minimum needed number of time slots in the higher order ODUk for accomodation of the client signal. The ODUflex container can easily be adapted to support higher or lower client signal bit rates simply by occupying a higher or lower number of 1.25 Gb/s slots in the ODU payload area.

3.3 TDM Multiplexing

The multiplexing structure in OTN is based on putting containers in containers, stepwise multiplexing the data to higher bit rates before putting it on a wavelength. An ODU can either be put directly into an OTU, or multiplexed with other ODUs to fit inside an OTU. Figure 3.3 illustrates how client signals are multiplexed and/or mapped into an OTU3 (40 Gb/s). A non-OTN client signal is firstly mapped into a lower order OPU, named OPU(L). The OPU(L) is mapped into the ODU(L), and further into the OTU[V]. OTN-signals are first mapped into the ODTU in various multiples depending on the bit rate (the ODTU is an ODU with justification overhead). The ODTUs are then multiplexed into an ODTUG. The ODTUG is mapped into a higher order OPU, named OPU(H). The OPU(H) is then mapped into an ODU(H) which is mapped into the OTU[V] [4]. The OPU(L) and OPU(H) have the same information structures, but with different client signals [4].

As seen from Figure 3.3; it is possible to either map a client signal directly or an ODTUG consiststing of interleaved lower order ODUs into the OPU payload area.

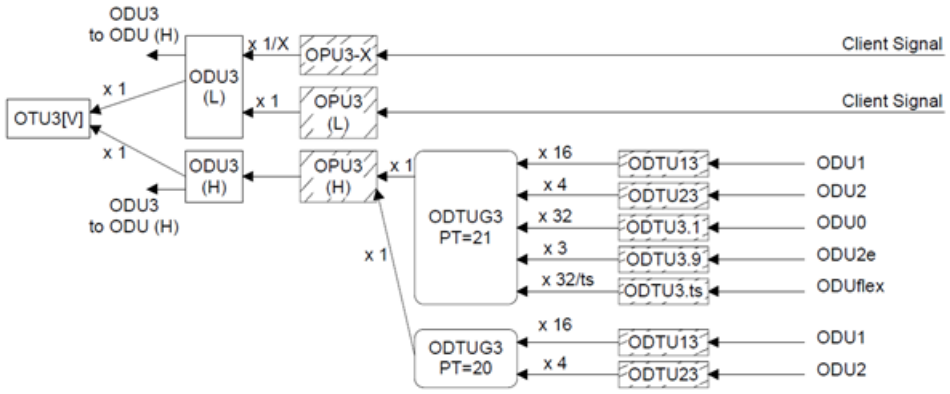

Figure 3.3: Example of the flexible multiplexing structure in OTN [4]

The ODUk (k = 0,1,2,3,4) is used as the basic multiplexing unit in OTN. Figure 3.3 shows only one of several multiplexing possibilities in the OTN hierarchy. It is for instance possible to multiplex two ODU0s into an OPU1, four ODU1s into an OPU2, four ODU2s into an OPU3 or eighty ODU0 into an OPU4.

3.3.1 Tributary slots

The overhead area of an ODUk consists of 16 columns and 4 rows, while the payload area consists of 3808 columns and 4 rows. The ODU payload area, which consists of OPU overhead and payload, is divided into tributary slots (time slots) in order to accomodate lower order ODUs. The tributary slots in the ODU payload area is assigned according to the required bandwidth for each client signal that is transported inside the OTN frame. The first tributary slot starts at column 17 (TS1), column 18 is used for tributary slot 2 etc. Column 3824 is used for TS4 in an OPU2 or TS16 in an OPU3 [15].

The slots are interleaved within the OPUk. Each slot includes part of the OPUk overhead area, and a part of the OPUk payload area. Two types of tributary slots are defined in the G.709 standard [4]:

-Tributary slot with a bandwidth of approximately 2.5 Gbit/s, the OPUk is divided into n tributary slots, numbered 1 to n
-Tributary slot with a bandwidth of approximately 1.25 Gbit/s, the OPUk is divided into 2n tributary slots, numbered 1 to 2n

The arrangement of tributary slots is done according to the multiplexing pattern.

An example of 1.25G tributary slot allocation in an OPU4 is seen in Figure 3.4. In this case, the OPU4 is divided into 80 1.25G tributary slots which are located in columns 17 to 3816 of the OTN frame. 8 columns of fixed stuff is located in columns 3817 to 3824. There are two options of presenting the OPU4 frame; a 320 row by 3810 column format or in a 160 row by 7620 column format (Figure 3.4 shows the 320 row by 3810 column format.) A 1.25G tributary slot occupies 1.247% of the OPU4 payload area. Each tributary slot has an associated tributary slot overhead area (TSOH). As seen from Figure 3.4; the TSOH for a 1.25G tributary slot is available once every 80 frames [4].

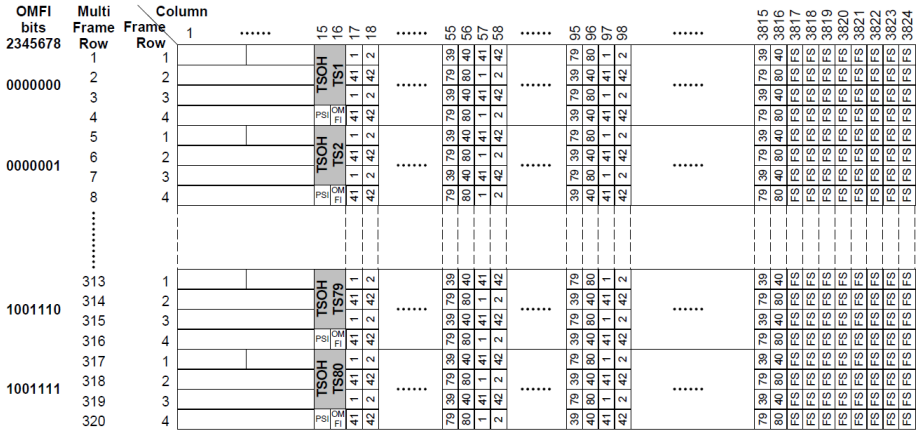

Figure 3.4: The figure shows allocation of OPU4 1.25G tributary slots in a 320 row by 3810 column frame format. 80 slots in columns 17 to 3816 of the OTN frame is used to accomodate lower order signals in the OPU4 frame [4].

3.4 OTN signal rates

Table 3.1 shows the various line rates used in OTN.

OTU type	OTU nominal bit rate
OTU-1	2.5 Gb/s
OTU-2	10 Gb/s
OTU-3	40 Gb/s
OTU-4	100 Gb/s

Table 3.1: The table shows OTN interface card line rates [4]. OTU-0 (1.25 Gb/s) is not included in the table as it isn't defined as an interface, but used as a multiplexing entity (ODU-0).

The line rates in OTN span from 2.5 (OTU-1) to 100 Gb/s (OTU-4). OTU-0 is not defined as an optical interface in OTN, but used as a basic multiplexing entity (ODU-0) [4].

3.5 OTN layers

The OTN layers (Optical Transport Hierarchy) are defined in the G.872 standard.

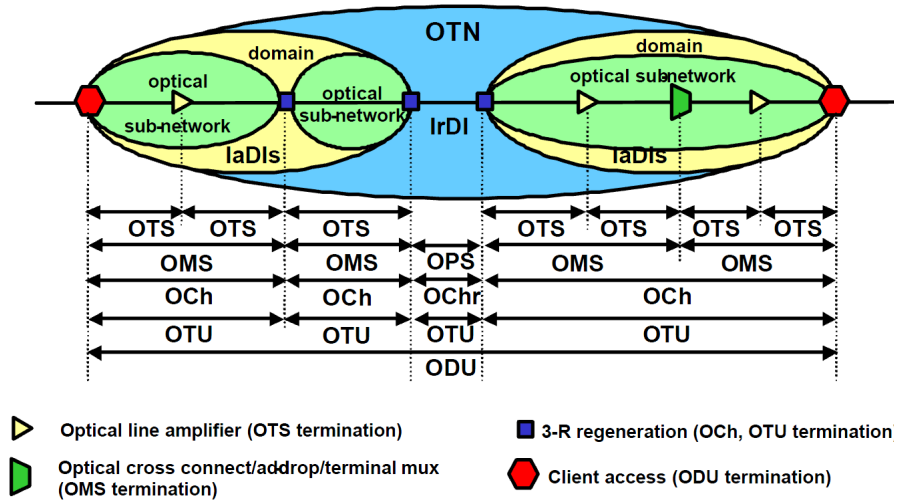

Figure 3.5: OTN layers [5]

Figure 3.5 presents the different layers in the OTN hierarchy. As seen from the figure; the ODU is terminated at OTN domain edges, the OTU and OCh are terminated at optical sub-network edges, the OMS is terminated in multiplexers/demultiplexers and OTS at optical line amplifier/regeneration points.

Figure 3.5 also shows the location of the Inter-domain interface (IrDi) and Intra-domain interfaces (IaDIs). The G.872 recommendation defines the IrDI as the location between the networks of two operators, between the sub-networks of two vendors in the same operator domain or the location within the sub-network of one vendor [18]. IrDI interfaces are defined with 3R processing at each end of the interface [5]. IaDI is defined as the location between the equipment of an individual manufacturer's sub-network [18].

3.6 FEC

Optical core networks transport signals over long spans reaching tens or hundreds of kilometers. Signals that are tranported over these distances are affected by severe impairments. The trend in long-haul optical transport networks is to use an increasing number of transparent all-optical network elements (OXCs, OADMs), without electrical/optical conversion [9]. While electrical components regenerates signals, the all-optical network elements transparently forwards signals without any signal regeneration.

An important and attractive feature with OTN is FEC. REED-Solomon code generates error-correcting bits that are added to the payload. Correction of bit errors that occur over the signal span can then be performed at the receiving end [5]. The FEC coding adds 6.7% overhead to the payload [4].

FEC can provide up to 6.2 dB increase in SNR [5]. In other words; a signal at a certain BER can be transmitted at 6.2 dB less power than possible without FEC. This SNR increase can be used to extend the span between generators, increase the number of channels in a DWDM system (which is limited by the output power of the amplifiers), and decrease the need for 3R regeneration [5].

3.7 TCM

TCM enables fault detection and monitoring between predifined sections in the network. It is possible to monitor signal quality spanning over several operator domains or sections between operator domains.

While SDH only allowed a single level of monitoring, OTN allows six levels of TCM to be defined independently [5]. TCM is manually configured by adding overhead to the ODU.

Figure 3.6 shows an example of nested and cascaded TCM connections.

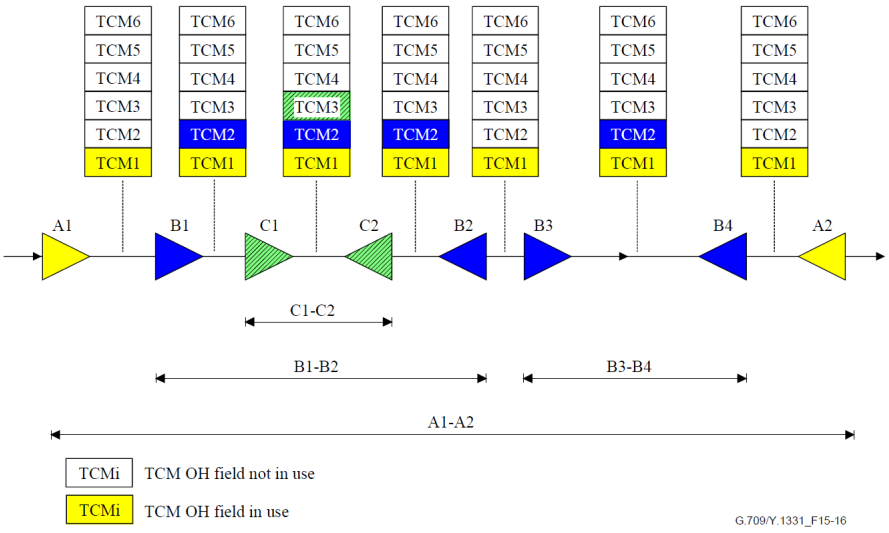

Figure 3.6: Example of nested and cascaded TCM connections [5]

Connections A1-A2/B1-B2/C1-C2 and A1-A2/B3-B4 are nested, while B1-B2/B3-B4 are cascaded.

3.8 Recent progress in OTN

OTN serves as the underlying technology for optical core networks, and must evolve as industry trends and requirements change. The set of recommendations that form the OTN standard is under constant development in order to support future demands for new services and higher bit rates.

The evolution of OTN is closely related to standardization activities in the IEEE on higher speed Ethernet, 40GbE and 100GbE. The maximum transmission distance in the Ethernet standards is 40 km, so a wide-area transport technology like OTN is necessary to support Ethernet connections above this distance [16].

The ITU-T Study Group 15 is responsible for the standardization activities in OTN. A new version of the G.709 recommendation was approved in December 2009 [19]. It had many additions compared to the former G.709 recommendation from 2003. No changes were however made that would impact networks and equipment built on the previous versions of the G.709 recommendation [20].

Important enhancements in the December 2009 G.709 release include [20]:

- **Specification of ODU0** The lowest multiplexing entity defined in the G.709 recommendation from 2003 was the ODU1, which corresponds to a bit rate of approximately 2.5 Gb/s. No ODU in the OTN hierarchy was tailored for transporting a single GbE (1.25 Gb/s) signal. The GbE signals had to be transported in the ODU1 containers, something which wasted approximately 50% of the capacity in the ODU1 payload area. The newly defined ODU0 has a bit rate of approximately 1.25 Gb/s and is perfectly suitable for transporting and monitoring Gigabit Ethernet connections.

- **Specification of a new transcoding mechanism for 40GbE transport** The bit rate of 40GbE (41.25 Gb/s) is greater than that of the ODU3 payload (40.15 Gb/s). A new transcoding mechanism was thus needed in order to accomodate 40GbE in an ODU3. Transcoding is the conversion of client signal coding, and was used to decrease the code redundancy. This resulted in a decrease in bit rate from 41.25 Gb/s to 40.12 Gb/s, enabling the possibility for transport of the 40GbE signal in the ODU3 payload.

- **Specification of ODU4/OTU4** After the standardization of 100GbE by the IEEE in June 2010 [21], it was desirable to transport this signal directly over OTN. ODU3 (40 Gb/s) provided the highest bit rate in the previous G.709 recommendation. This led to the specification of the

ODU4/OTU4. The bit rate of the ODU4 is slightly higher than what is needed to transport 100GbE (the ODU4 payload capacity is 104.356 Gb/s). This was done so that it can carry up to 10 x ODU2e signals. (The ODU2e was standardized in 2006 as a solution for transporting 10GbE over OTN, because the bit rate of 10GbE (LAN PHY: 10.3125 Gb/s) is greater than the payload capacity of ODU2 (9.99528 Gb/s) [16]. The bit rate of ODU2e is approximately 10.3995 Gb/s.) All the previously defined ODUs can be multiplexed into the ODU4.

- **New size of tributary slots** Prior to the release of the newest G.709 recommendation, the tributary slots had a size of 2,5 Gb/s. After the release of the newest recommendation, the tributary slot size is also defined at 1,25 Gb/s granularity.

- **Specification of ODUflex** As previously explained; the ODUflex container is used to give support for future client bit rates. This is done by occupying a higher or lower number of 1.25Gb/s tributary slots in the ODU payload area.

The highest client signal rate currently supported by OTN is the ODU4/OTU4 which has a bit rate slightly higher than 100 Gb/s. IEEE is currently discussing the next higher speed Ethernet standard. This might be 400GbE, 1TbE or possibly higher rates. ODU5/OTU5 is being considered by the ITU in order to support higher client signal bit rates. The bit rate of the ODU5/OTU5 will be decided on basis of the future client signal rates.

Chapter 4

OTN switching

4.1 Background

Switching technology used in the optical core network gets increasingly important as router interconnection speeds continuously reaches higher levels.

A lot of research is currently carried out on all-optical packet switching. With optical packet switching, the packet stays in the optical domain from network ingress to network egress. This is very attractive from a performance point of view, but many obstacles are still to overcome before these types of switches will be seen in real life action. The main challenge is the buffering of packets (by the use of e.g. fiber delay lines) in order to avoid contention in the switch [22].

Currently available all-optical switching is performed by optical cross connects. OXCs offer full transparency to protocol format and bit rate, but is only able to perform switching at wavelength granularity. Electrical switching (the signal is converted from optical to electrical, switched, and reconverted to optical) is still the only way of performing switching at sub-lamda level.

SDH and SDH-based switches were originally designed for transporting/switching voice traffic. There has been a shift from transporting voice traffic to transporting data traffic in the core networks since SDH was developed in the 80's. Today, almost all traffic transiting the core network is data [3]. Data traffic differs largely from voice both in transmission rates and format. Switching must now take place at much higher bit rates than previously required.

It is preferable to perform OTN switching in network nodes if OTN is used as transport technology. OTN switching offers a flexible form of electronic switching, and the full potential of OTN is first unleashed when OTN switching is used in the nodes. The flexible TDM multiplexing hierarchy in

OTN is based on the organization of data in ODUs. The main benefit of this multiplexing structure is that lower order ODUs that are mapped into higher order ODUs remains individually switchable. This allows for switching at ODU bit rate granularities.

Almost 70% of all traffic that arrives at a node in the core network is through-passing traffic (destination is another node) [3]. IP routers are forced to unnecessarily process all through-passing traffic if no bypass capabilites are available. IP routing works by examining the header of each incoming packet. The addressing information contained in the header is matched with the routing table in the IP router, and the next hop is determined. The packet is then copied to the correct outgoing interface. This process is resource consuming. If bypass of intermediate IP routers is impossible, the router processing resources are unnecessarily wasted. The routers must be more powerful, and thus more costly than what is actually needed. A sufficient increase in bypass traffic will at the same time require an increase in IP router capacity. Routers that can scale to terabit capacities have been developed. The cost and complexity of such routers are however very high. Bypass of IP routers is thus increasingly important as the core traffic volume increases.

OTN switching offers effective bypass of intermediate IP routers by switching bulks of data at layer 1. A higher level of throughput, and a more cost-efficient network architecture is probably achieved if data is switched on a coarse granularity instead of routed on a per-packet basis.

OTN offers switching at the different ODU granularities that the OTN multiplexing hierarchy offers; 1.25, 2.5, 10, 40, and 100 Gb/s. Traffic in the core network is highly aggregated, and big amounts of traffic are usually following the same path between central nodes, destined for the same edge node in the network. Switching traffic towards its destination in coarse bulks of data is thus a much more effective way of forwarding traffic, than performing routing on a per-packet basis.

Lambda switching performed with optical crossconnects lacks grooming and degrooming capabilities. OTN switching, on the other hand, has the ability to combine (groom) and split (degroom) signals from various flows in OTN switching nodes. It combines the speed of low layer switching with the features of layer-3 routing, achieving high node throughput by coarse granularity switching and router bypass.

It is also possible to combine OTN switches with optical crossconnects. This makes it possible to establish direct connections between OTN switching nodes, bypassing intermediate nodes in the optical layer, and at the same time providing high wavelength utilization.

4.2 Node architecture

There are two possible approaches to switching with OTN. One approach is at OCh-level performed in optical cross connects. This approach implies switching at wavelength granularity in the optical layer (lambda switching). If wavelength switching is used, it is not possible to switch at lower granularities (sub-lambda) than wavelength granularity. The switching rate is thus bound to the signal rate. The signal is switched all-optical and is fully transparent. No conversion to the electrical domain is needed to perform this type of switching. Keeping the signal in the optical domain implies no signal regeneration, and performance monitoring is not possible.

The other approach is electrical switching at ODU level. This approach is needed to add/drop lower speed streams to/from a higher speed multiplex (sub-lambda switching). This feature is needed in order to provide higher efficiency in filling of the optical channels and in order to de-couple the topology of the ODU connections from the topology of the OCh connections [23]. Regeneration of the optical signal is performed in the electrical switch, and signal performance monitoring is possible. Electrical switching is opaque because of its dependence of client signal type, bit rate and modulation format. The electrical switch can, in contrast to the optical switch, only switch supported client signal types.

The proposed switch is an opaque electrical switch, which performs switching at ODU level. The client interface cards thus limits the supported input signals.

Figure 4.1 illustrates how the OTN switch is located between the optical crossconnect and IP router in the network node. OTN switching performs, as illustrated in the figure, "medium coarse" granularity switching. The OTN switch switches ODUs at ODU bit rate granularities instead of switching single packets (IP router), or whole wavelengths (optical crossconnect).

Figure 4.1 also illustrates how a group of incoming wavelength division multiplexed signals enter the node (OCG is the OTN term for a group of multiplexed OCCs). The OCG is demultiplexed to OCCs (individual wavelengths) and enters the optical crossconnect. The optical crossconnect is configured to cross connect wavelengths arriving at certain input ports to certain output ports. Wavelengths which transport data that should be added and/or dropped in the current network node is cross connected to the OTN switch. The OTN switch receives/transmits optical signals, converts them to/from the electrical domain, adds/extracts Ethernet frames to/from ODUs, and performs switching to the correct output.

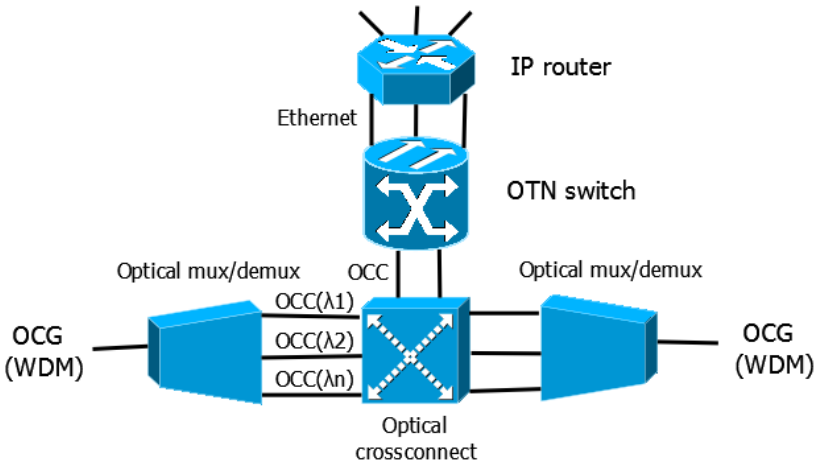

Figure 4.1: OTN node architecture. The figure shows how the OTN switch is located between the optical crossconnect and IP router.

The OTN switch improves filling of the wavelengths by multiplexing lower rate streams into higher rate streams. Using an optical crossconnect in combination with the OTN switch makes it possible to bypass the network node in the optical layer (optical bypass). The OTN switch also makes sure that only IP traffic that is destined for the current node is dropped. In this way, unnecessary processing of bypass traffic in the IP routers is avoided.

4.3 OTN switch proposal

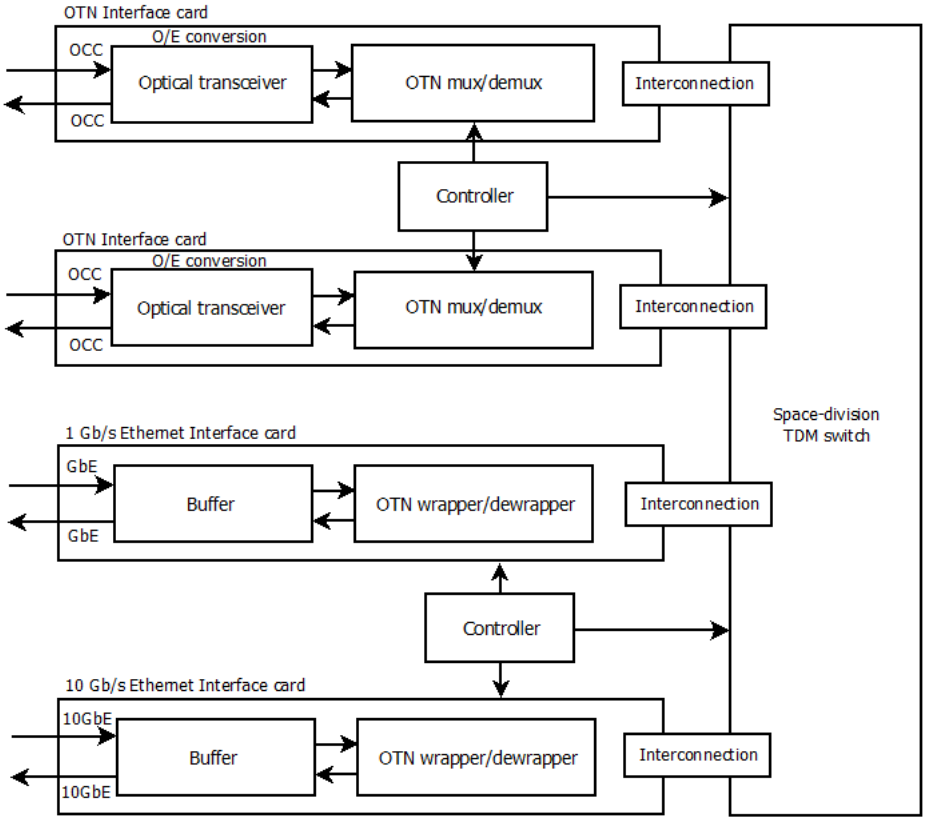

Figure 4.2: OTN switch proposal. Interface cards are seen in the left part of the figure. The TDM space-division crossbar switch in the rightmost part of the figure is used for switching ODUs/Ethernet frames between the different interface cards.

The main parts of the proposed OTN switch seen in Figure 4.2 are client interface cards and a TDM space-division crossbar switch. The client interface cards are responsible for receiving and sending various client signals. It is possible to add other types of client interface cards as the switch is made in a modular design. Other types of client interface cards that could be added to the switch is for instance SDH and ATM line cards. This would make it possible to add/drop SDH and ATM signals to/from OTN frames in the transport network. With the proposed OTN switch, it is possible to add and/or drop Ethernet frames to/from higher-order OTN frames in the optical transport network.

The OTN and Ethernet interface cards as well as the TDM space-division crossbar switch that are part of the proposed OTN switch are explained in the three following sections.

4.3.1 OTN interface cards

The OTN interface cards are located in the top left of the proposed switch in Figure 4.2. These cards are responsible for receiving and transmitting OTN signals which are modulated onto a wavelength. The interface cards accept an incoming OCC (wavelength) in the optical transceiver. The optical transceiver is responsible for converting optical signals into electrical, and vice versa. The OCC is converted to the electrical domain (OTU-n) in the transceiver, and the OTU-n is forwarded to the OTN mux/demux. The OTU-n is demultiplexed into the ODTUs that form the OTU (ODTUs are ODUs with extension overhead) in the OTN mux/demux. The extension overhead is removed, and the ODUs are now individually switchable. The controller keeps track of the individual ODUs. The TDM space-division crossbar switch is used to switch the ODUs between the different interface cards.

4.3.2 Ethernet interface cards

The Ethernet interface cards makes it possible to add and drop 1 and 10 Gb/s Ethernet client signals to/from the OTN switch. Incoming Ethernet frames are first sent through the buffer in the interface card. After passing the buffer, the frame is handled in the OTN wrapping unit. The OTN wrapping unit is responsible for extracting Ethernet frames from the ODU (Ethernet traffic that is dropped from the switch), or wrap Ethernet frames into an OPU and further into an ODU (Ethernet traffic that is added to the switch). This ODU can be switched to the desired OTN interface card, and multiplexed to a higher order OTU together with other ODUs in the OTN muxing unit.

4.3.3 TDM space-division crossbar switch

A possible implementation of a TDM space-division crossbar switch taken from [6] is seen in Figure 4.3.

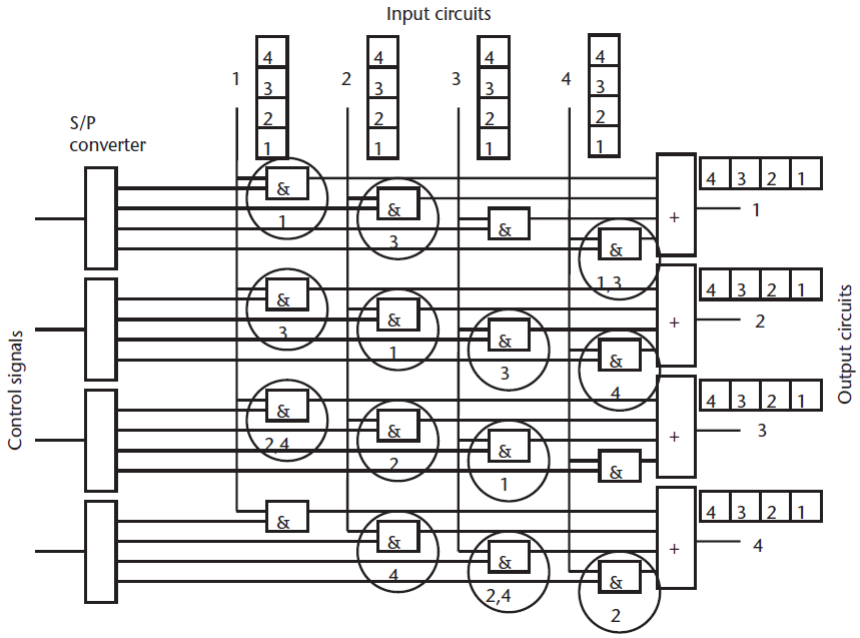

Figure 4.3: TDM space-division switch capable of switching ODUs between timeslots [6].

The switch consists of input, output and control circuits in addition to the crossbar matrix. The control signals are used to control which time slot of the input circuit that is passed to a certain time slot of the output circuit.

The TDM space-division switch makes it possible to switch individual ODUs between timeslots and interface cards.

Chapter 5

Network scenario

This chapter presents a basic network scenario that will be realized using IP packet switching and OTN switching. The goal of the network scenarios, and subsequent simulator implementations, is to evaluate the potential benefit of performing OTN switching in the core network. Simulation results will be used for comparing the performance of IP packet switching with OTN switching.

5.1 Basic three-node network scenario

The basic network scenario is seen in Figure 5.1, and consists of three nodes. The leftmost node is an aggregation node which aggregates incoming traffic from several interfaces. The middlemost node (router 2) has three client interfaces. Router 3 has seven client interfaces, and drops all traffic that is received to its client interfaces. As previously mentioned; almost 70% of the traffic that arrives at a node in the core network is through-passing traffic [3]. Based on this fact, 30% of the incoming traffic is on average dropped to the client interfaces in router 2, while 70% of the traffic is forwarded to router 3. The three routers are interconnected with optical fiber.

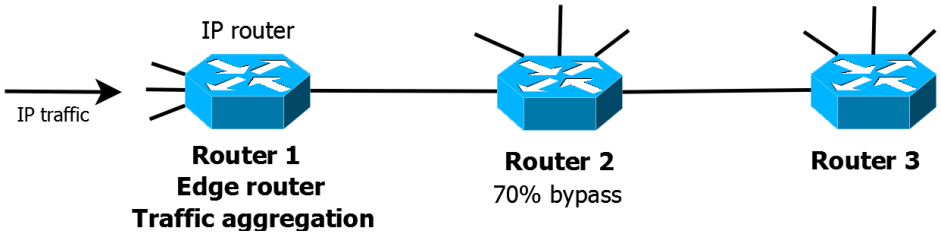

Figure 5.1: Three-node basic network scenario

5.2 Three-node packet switching scenario

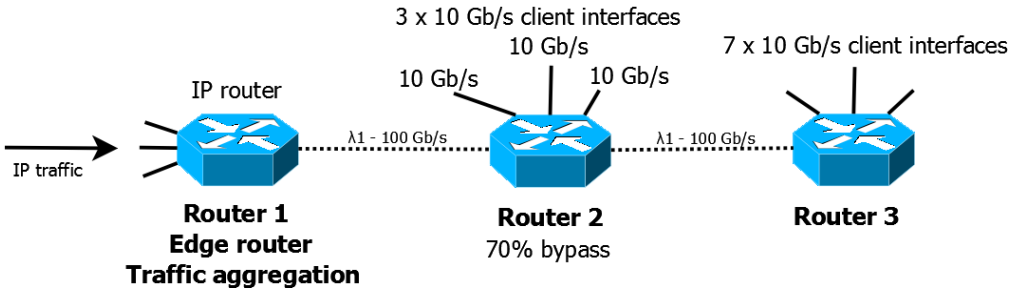

Figure 5.2: Three-node packet switching scenario

The three-node packet switching scenario is a realization of the basic network scenario and is seen in Figure 5.2. The fiber is connected directly between the routers, and a single wavelength of 100 Gb/s is used in each fiber. This is also known as a "big fat pipe". The 100 Gb/s wavelength capacity is fully shared for all traffic. There is no way of bypassing traffic in router 2. All traffic destined to router 3 is processed by router 2 before being forwarded, and is dropped to the client interfaces when arriving at router 3. Router 2 has three 10 Gb/s client interfaces, while router 3 has seven 10 Gb/s client interfaces. 30% of the traffic is on average dropped to the interfaces in router 2, while 70% on average is forwarded to router 3.

5.3 Three-node OTN switching scenario

The three-node OTN switching scenario is seen in Figure 5.3.

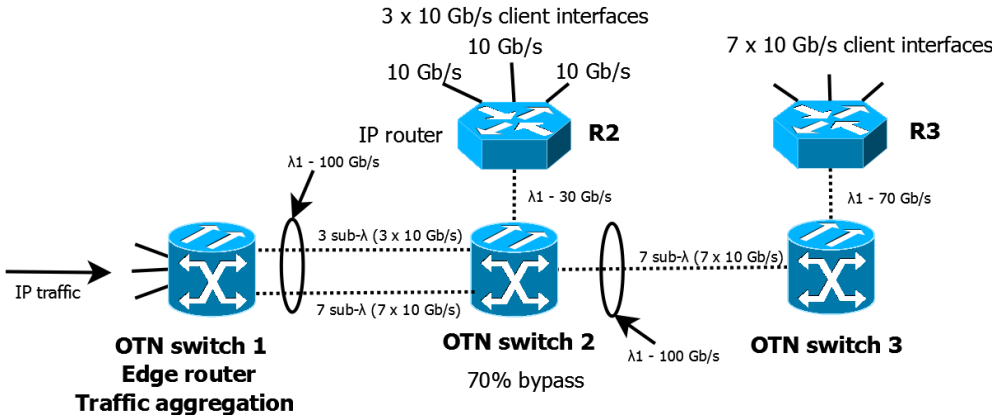

Figure 5.3: Three-node OTN switching scenario

The OTN switching scenario is the basic network scenario "solved" with OTN switches in addition to IP routers.

As seen from the figure; the main difference from the packet switching scenario is the division into sub-lambda granularities. By using the TDM hierarchy offered by OTN, it is possible to divide the wavelength bit rate (100 Gb/s) into sub-bit rates. Three sub-lambdas (3 x 10 Gb/s) are assigned to router 2, while seven sub-lambdas (7 x 10 Gb/s) are assigned to router 3. In this way, the network is statically configured for the expected network load. OTN switch 2 only drops traffic that is destined for router 2 to its Ethernet client interfaces, while traffic destined for router 3 is forwarded by the OTN switch. Traffic destined for router 3 is thus bypassed, avoiding processing in router 2.

As with the packet switching scenario; 30% of the traffic is on average destined to router 2, while 70% on average is destined to router 3.

Chapter 6

Simulation model

A simulation model has been developed for each of the two previously described network scenarios (packet switched and OTN switched three-node scenario). The simulators have been developed using the SIMULA programming language. The DEMOS context class is used to enable the use of discrete events.

This chapter explains the implementation of the simulation models. The next chapter presents and discusses results obtained from the simulation runs.

6.1 Background

Simulation is a simple and flexible way of analyzing systems. The simulator can be designed to include exactly the needed level of details to give a good representation of the real system [24].

A discrete-event system is a system which behavior is governed by events occuring at discrete points in time and resulting in distinct changes of the state of the system [24]. In this case, the discrete events are packet arrivals/departures.

When developing a simulator that is based on a given scenario, some simplifications must be made. In this thesis, the goal for the simulation runs is to evaluate the difference in queueing systems experienced when using a pure packet switched network and an OTN switching based network. Processing/switching delay in routers/switches is thus neglected in order to get simulation results that reflect the difference between the alternative queueing systems.

6.2 Common simulator implementation parts

Simulator implementation parts that are used both in the IP and OTN simulators are as follows:

Packet generator The packet generator is scheduled at simulation start, and is responsible for generating IP packets. Each IP packet that is generated draws a sample from a packet size distribution in order to determine the packet size.

Packet size distribution CAIDA gathers statistics about packet sizes in the Internet. Traffic investigations made by CAIDA show that there are a predominance of small packets, with peaks at the common sizes of 44, 552, 576 and 1500 bytes in the Internet [7]. From [7], we find the cummulative distribution of packet sizes in the Internet. The distribution is seen in Table 6.1.

Packet length	**Cummulative percent**
40	0.00
44	0.62
552	0.75
576	0.83
1500	1.00

Table 6.1: Empirical distribution of IP packet sizes in the Internet[7]

This distribution is used for setting packet lengths in the simulator. The size of each generated packet is determined by pulling a sample from the packet size distribution. The average packet size is 286,36 bytes, and the maximum packet size is limited to 1500 bytes.

Packet arrival process The packet generator holds for an interarrival time between the creation of each packet. Two possible packet arrival processes are implemented for the simulators; a negative exponential interarrival distribution and a H_2 hyperexponential interarrival distribution.

When the N.E.D. distribution is used to control the interarrival of packets, a sample is drawn from the distribution, and the simulator holds for the duration of this sample between the creation of packets. A mean value is inputted into the distribution in order to control the average load on the links.

The hyperexponential interarrival distribution is implemented in order to reflect the possible burstiness in the Internet. The distribution is realized by using a combination of two negative exponential distributions. One distribution is set to generate a normal traffic load, while the other is put to a much higher expected arrival rate (e.g. 50 times higher than the normal) in order to create burstiness in the traffic pattern at moments in time. After the generation of a packet, the interarrival distribution with low interarrival time (bursty traffic) is chosen with a probability of A, while the distribution with normal interarrival times is chosen with a probability of 1-A. This generates in total a much more bursty traffic pattern than the single N.E.D. interarrival distribution.

A counter named "generatedPackets" is increased with one after the creation of each packet.

Measuring packet delay and loss Each packet that is generated is timestamped with the current clock time (timestamp 1). If the packet reaches its destination (gets dropped to the client interfaces in router 2 or 3), the packet gets a second timestamp (timestamp 2). The packet delay is found by extracting timestamp 1 from timestamp 2. The counter "receivedPackets" is increased by 1 when the packet successfully reaches its destination.

If a packet arrives at a full buffer, the packet is dropped (lost). The counter "droppedPackets" is then increased by one. In this case, the delay is not calculated.

The statistics of all three counters (generatedPackets, receivedPackets and droppedPackets) are printed in the trace file at simulation end.

Simulation parameters Four different parameters are important in order to obtain good statistical results from the simulation runs. These are transient simtime, holdtime, number of replications and simulation seed.

The transient simtime is used to make sure that the simulator is in steady state before statistics are logged. The holdtime controls the length of the simulation run, and must be long enough to provide trustworthy results. Number of replications controls the number of individual simulation runs that should be run to create reasonable statistical variance. The simulation seed can be put to any integer, and provides an arbitrary simulation seed to each replication.

6.3 IP simulator

The IP simulation model is seen in Figure 6.1. The IP simulator is based on the three-node packet switching scenario presented in the previous chapter. An edge router aggregates traffic and forwards all incoming packets into the first shared output queue (router_out_Q1). A single wavelength with a bit rate of 100 Gb/s is used for transmission between the routers. Router 2 drops 30% of the incoming traffic to its client interfaces, while 70% of the traffic is forwarded to a second shared output queue (router_out_Q2). All forwarded traffic is dropped to the client interfaces at router 3.

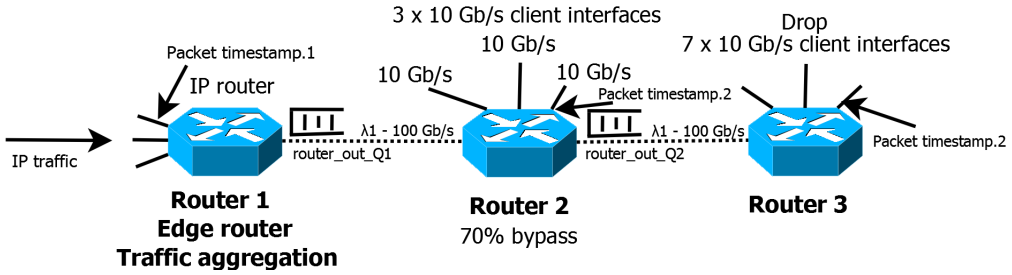

Figure 6.1: IP simulation model. The simulation model is the "DEMOS view" of the three-node packet switching scenario.

Implementation parts that are special for the IP simulator are as follows:

Queues and transmission The pure IP simulator uses shared output queues on the routers. The queues are modelled by using the inbuilt waitq entity in DEMOS. The maximum time for a packet to be buffered in a router should be maximum 200 ms. The output queues in the simulator have a buffer size of 12,5 kilobytes, providing capacity for an average of 43 packets in a full buffer. A packet that enters a buffer with one free slot (42 packets already in buffer) will not experience a bigger delay than 200 ms.

The transmission delay is simulated by holding the simulator for the packet length divided by link capacity for each packet that leaves the output queue. The link capacity is scaled down in the simulator because of length limitations in the DEMOS variables. The 100 Gb/s links connecting routers 1, 2 and 3 is thus each set to a capacity of 10^6 b/s.

When a packet arrives at router 2, a random number between 1 and 10 is drawn. If this number is bigger than 3 (70%), the packet is forwarded to router 3 (put in output queue 2). If the number is smaller than 3 (30%),

the packet is dropped to the client interfaces in router 2, and the counter "receivedPackets" is increased by 1.

6.4 OTN simulator

The OTN simulator is an implementation of the three-node OTN switching scenario presented in the previous chapter.

The OTN simulator models the impact of using OTN switches combined with IP routers in the core network. As previously explained; OTN switches makes it possible to perform sub-lambda switching, meaning that it is possible to switch individual ODUs at sub-wavelength bit rate. Switching at sub-wavelength bit rate makes it possible to drop or bypass traffic in the OTN switches depending on whether packets contained in the OTN frame should be forwarded to the IP router in the network node or not.

The DEMOS implementation of the three-node OTN switching scenario is seen in Figure 6.2.

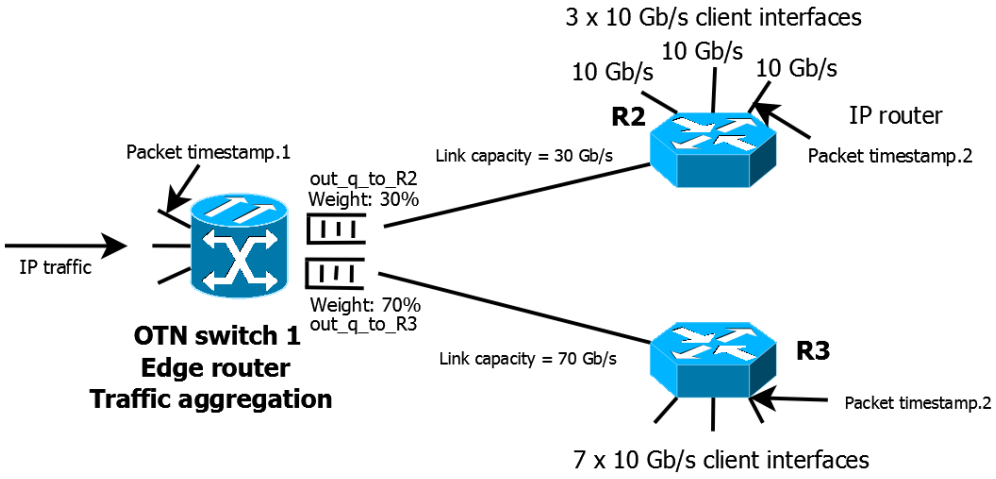

Figure 6.2: OTN simulation model. The simulation model is the "DEMOS view" of the three-node OTN switching scenario.

The implementation of the simulator in terms of packet generator, packet size distribution and measurement of packet delay and loss is similar to the IP simulator. The difference is in how the queues and links are configured.

Queues and transmission An edge router (OTN switch) aggregates traffic and forwards all incoming packets in the OTN simulation model.

As explained in chapter 3 (G.709 chapter); the OPUk area in the OTN frame is divided into a number of tributary slots (time slots) in order to share the bandwidth amongst different streams. If the edge router was a real OTN switch it would accept IP traffic on several input interfaces and bundle this into OTN frames which would be sent on a single wavelength. In the OTN simulator, the packets are shared amongst two separate output queues in order to simulate the behaviour of the OTN switch (sharing bandwidth by the use of timeslots and bypassing intermediate nodes). Each output queue has its own router destination. Out_q_to_R2 sends all traffic to router 2, while out_q_to_R3 sends all traffic to router 3. An individual link is connected to each of the output queues. The link connecting the aggregation switch and router 2 has a capacity of 30 Gb/s, while the link connecting the aggregation switch and router 3 has a capacity of 70 Gb/s. In the simulator, the 30 Gb/s link is set to $0.3*10^6$ b/s and the 70 Gb/s link is set to $0.7*10^6$ b/s.

The incoming IP packets at the aggregation switch is tagged with a destination node and is put directly in the output queue associated with the destination node. 30% of the traffic arriving at the aggregation switch is on average put in out_q_to_R2 (destined to router 2), while 70% of the traffic on average is put in out_q_to_R3 (destined to router 3). As with the IP simulator; the traffic is put in the two different queues based on sampling of a random number between 1 and 10. If the number is below 3, the packet is put in output queue to R2. If the number is above 3, it is put in output queue to R3.

The output queues are, as with the IP simulator, also modelled with the waitq entity in DEMOS. The same buffer size (12500 bytes/average 43 packets) is used for each queue.

6.5 Reconfigurable OTN simulator

The reconfigurable OTN simulator is, in contrast to the regular OTN simulator, capable of reconfiguring the link capacities according to the traffic arrival intensity to the two output buffers. The switch measures the current queue lengths in both output buffers for each packet that is generated, and adjusts the link capacities based on the measurements. If the length of one output queue is longer than the other, then the link capacity of the link connected to the output buffer with higher queue length is increased, and the capacity of the link connected to the output buffer with shorter queue length is decreased. In this way, the capacity is dynamically adjusted based on the current traffic pattern.

The DEMOS implementation of the reconfigurable OTN switch is seen in Figure 6.3.

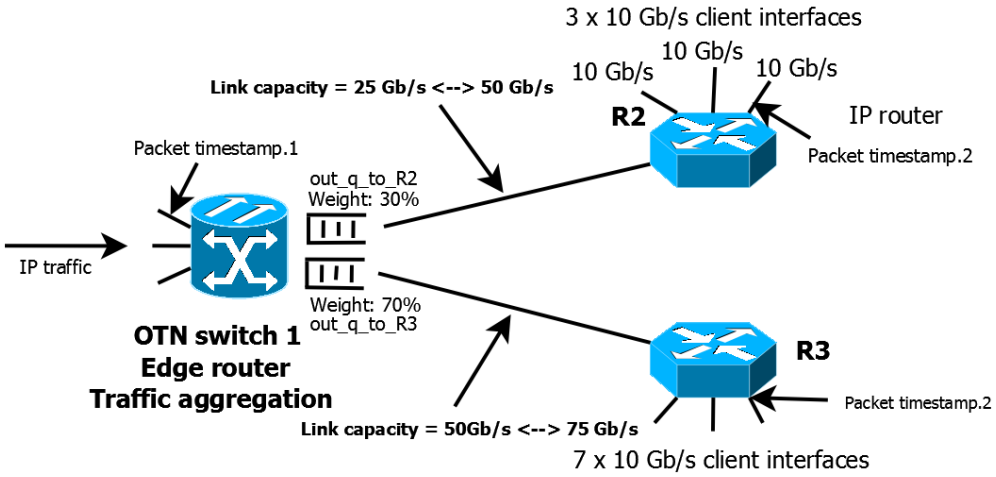

Figure 6.3: Reconfigurable OTN simulation model.

As seen from the figure; the reconfigurable OTN simulation model is similar to the regular OTN simulation model, but the link capacities are automatically reconfigurable. Each of the link capacities can be adjusted with +/- 1 Gb/s for each packet that is generated based on the current measured queue lengths. The link connecting output buffer out_q_to_R2 and R2 can be adjusted between 25 and 50 Gb/s, while the link connecting output buffer out_q_to_R3 and R3 can be adjusted between 50 and 75 Gb/s. The distribution of traffic amongst the two output queues is still the same; 30% of the traffic is on average put in output queue to R2, and 70% on average in output queue to R3.

Chapter 7

Simulation results

This chapter firstly presents the configuration of the simulation runs with the previously presented IP and OTN simulators. Secondly, the results from the simulation runs are presented. Parameters that are used for performance evaluation are average packet loss and average packet delay.

All simulators are first run with the negative exponential packet interarrival distribution with increasing network load. This is considered to be a distribution with low level of traffic burstiness. The second run is carried out with the hyperexponential packet interarrival distribution with increasing network load. This distribution generates a much higher level of burstiness in the traffic patterns, thus generating dense bulks of traffic in periods of time.

Lastly, the results are compared in common graphs. Each simulation run is replicated 10 times with individual seeds. The results are plotted with 95% confidence interval with 9 degrees of freedom using the Student's t-distribution.

7.1 IP simulator

The IP simulator is set up with simulation parameters seen from Table 7.1.

Parameter	Value
Holdtime	10000
Transient simtime	2000
Number of replications	10
Simseed	30
Buffer size (bytes)	12500

Table 7.1: Simulation parameters, IP simulator

The holdtime, which controls the duration of the simulation is put to 10000. This is a level which is long enough to provide reasonable results, but at the same time short so that the simulation runs are practically feasible to carry out. The transient simtime is put to 2000. This parameter makes sure that the simulator is in steady state (transient period has ended) before statistics are logged. 10 replications, each with different seeds are run to obtain results with good statistical properties. The simulation seed can be put to any valid integer. The simseed parameter is thus put to 30. The buffersize of the two output queues is each put to 12,5 kilobytes. The average packet length in the simulator is 286,36 bytes. The buffer has thus on average room for 12500/286,36 bytes = 43 IP packets.

7.1.1 N.E.D. interarrival distribution

The IP simulator is first run with the negative exponential interarrival distribution.

Delay Figure 7.1 shows the average packet delay with increasing average link load and N.E.D. interarrivals from the IP simulator. The increasing network load (A, offered traffic) is plotted along the x-axis, while the average packet delay (ms, milliseconds) is plotted along the y-axis.

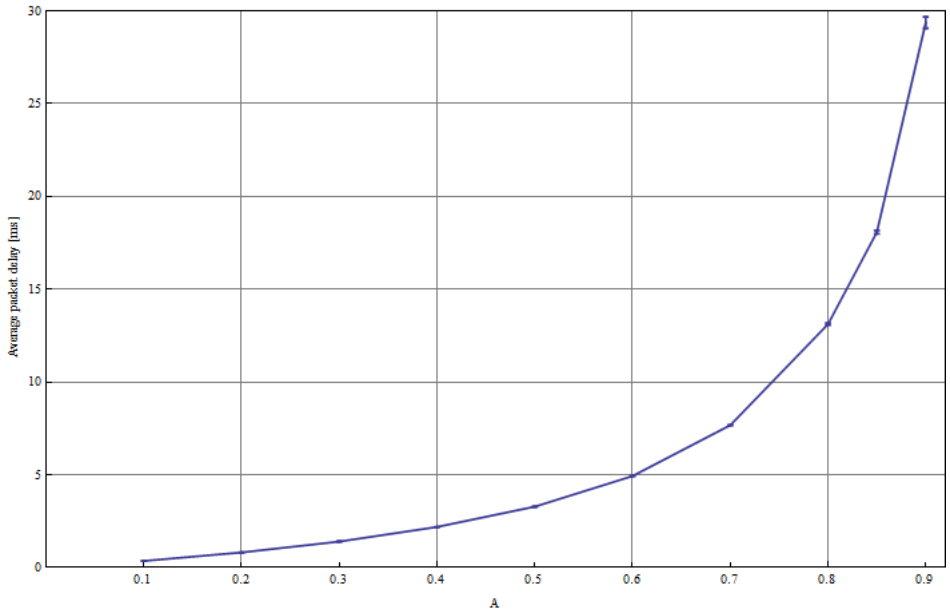

Figure 7.1: Average packet delay with increasing traffic load - IP simulator, N.E.D. interarrival distribution

The simulator is run with the average link load increasing in steps from 0.1 to 0.9 Erlang. As seen from the graph; the average packet delay increases with increasing link load. The average packet delay is 0.365 ms at 0.1 Erlang, and 29.37 ms at 0.9 Erlang.

The links connecting router 1, 2 and 3 are shared between all packets when they are sent between routers in the IP scenario. The increase in delay is due to the increasing number of packets sharing the capacity offered by the links. Few packets share the transmission capacity when the load is low, and filling of the output buffers is thus also low. The packets have low queuing delay when the output buffer filling is low, and are efficiently transmitted from router to router without spending much time in the queue. A heavy increase in delay is experienced in the area above 0.7 Erlang. This is because the output buffers reaches its maximum capacity and starts dropping packets when the network load reaches this level. When the buffers on average contain more packets (are full), the average packet delay also increases.

Packet loss Figure 7.2 shows the average packet loss ratio (generated packets/average dropped packets) with increasing network load. The network load ranging from 0.7 to 0.9 Erlang is plotted along the x-axis, while the average packet loss ratio is plotted along the y-axis.

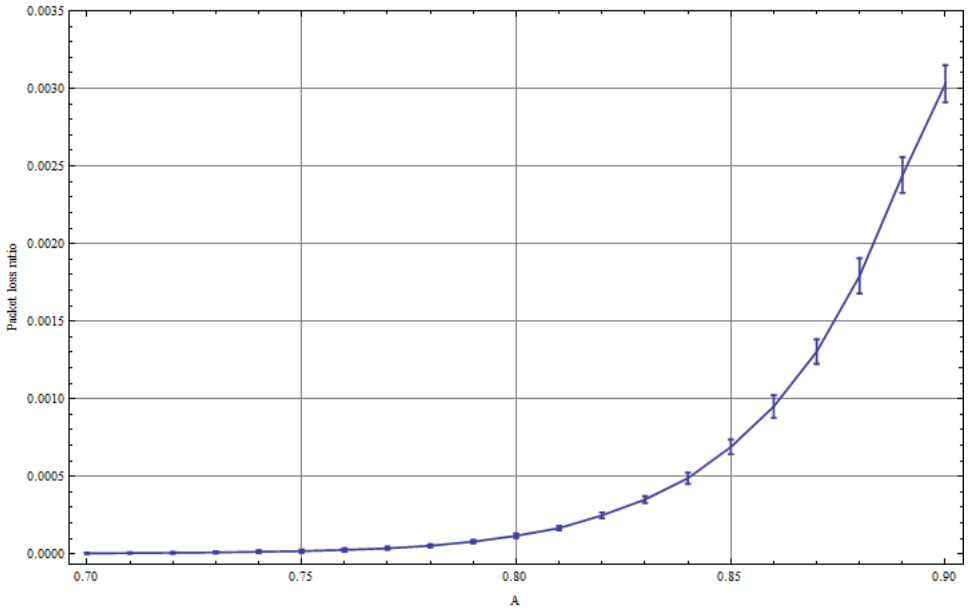

Figure 7.2: Average packet loss with increasing traffic load - IP simulator, N.E.D. interarrival distribution

As seen from the graph; packet loss starts occuring above 0.7 Erlang, and increases heavily as the load approaches 0.9 Erlang. The experienced packet loss in the range from 0.70 to 0.75 Erlang is very low. The packet loss ratio is 5.24E-6 (average 12.8 dropped packets of 2444000 generated packets) at 0.70 Erlang, and 1.25E-5 (average 32.2 dropped packets of 2619000 generated packets) at 0.75 Erlang. The loss in this area is thus just barely visible on the graph. The packet loss ratio increases heavily from 0.8 to 0.9 Erlang.

Packet loss occurs because of full buffers. The buffers get full when the traffic arrival rate to the buffer exceeds the departure rate from the buffer. As seen from the graph; 0.7 Erlang is the "breaking point" where the arrival rate to the buffer exceeds the departure rate, and packet loss starts occuring. The breaking point at 0.7 Erlang was also experienced with the average packet delay, which had a steep increase above 0.7 Erlang. This is because the output buffers on average are full at 0.7 Erlang and above.

Increasing the buffersize will make the system more loss tolerant (a higher traffic load is needed before packet loss occurs), but the average packet delay will increase. This is because the packets then on average will experience a higher waiting time in the buffer. Reducing the buffersize works in the opposite way; packet loss is increased and packet delay decreased.

7.1.2 Hyperexponential interarrival distribution

The hyperexponential distribution is now used for controlling the interarrival of packets. Samples from the interarrival distribution is used to decide for how long the simulator should hold before a new packet is generated. As previously explained; the hyperexponential distribution generates more burstiness in the traffic arrival pattern compared to the N.E.D. distribution. Groups of interarrival samples with low interarrival times are experienced when the hyperexponential interarrival distribution is used.

Based on parameters from [25], the interarrival intensities is put so that $\lambda 2 = 50\ \lambda 1$, and A = 0.2. When A is put to 0.2, the distribution with high traffic intensity is picked with a probability of 0.2, while the distribution with low intensity is picked with a probability of 0.8. The average link load is controlled by adjusting the expected interarrival times of the two distributions proportionally. $\lambda 2$ is constantly kept 50 times bigger than $\lambda 1$. The traffic pattern will now contain bursts of packets in periods of time during the simulation runs.

Delay Figure 7.3 shows the average packet delay with hyperexponential interarrival distribution from the IP simulator.

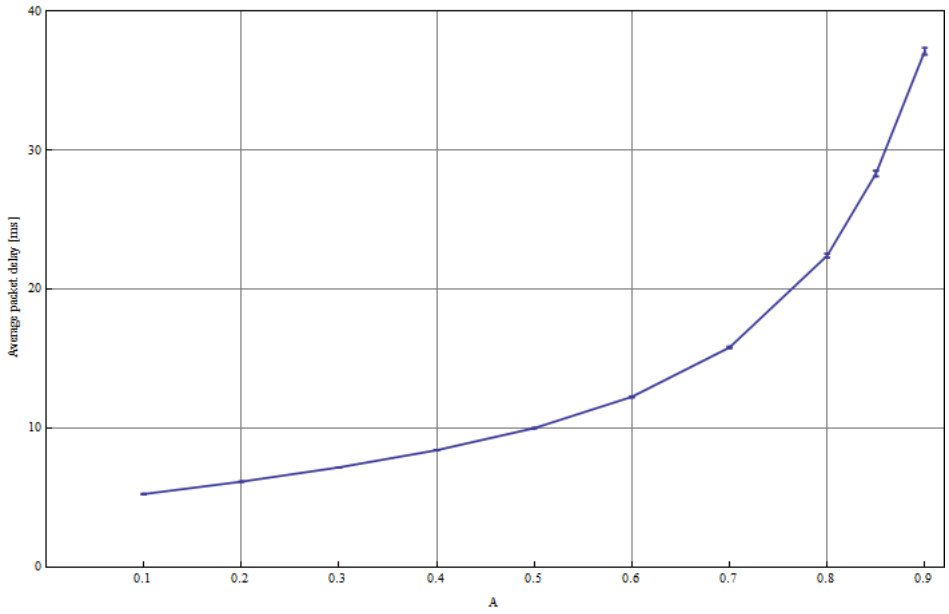

Figure 7.3: Average packet delay with increasing traffic load - IP simulator, Hyperexponential interarrival distribution

The average packet delay is higher with the hyperexponential interarrival distribution compared to what is experienced with the N.E.D. interarrival distribution. The average packet delay is 5.26 ms at 0.1 Erlang, and 37.1 ms at 0.9 Erlang. This is because of the bursty traffic pattern. Bursts of packets makes the filling of the buffers on average higher than without the bursty pattern. The packets are thus experiencing higher average waiting times in queue. It is also observed from the graph that the average packet delay, with this interarrival distribution as well, increases more heavily above 0.7 Erlang.

Packet loss Figure 7.4 shows the average packet loss ratio with hyperexponential interarrival distribution.

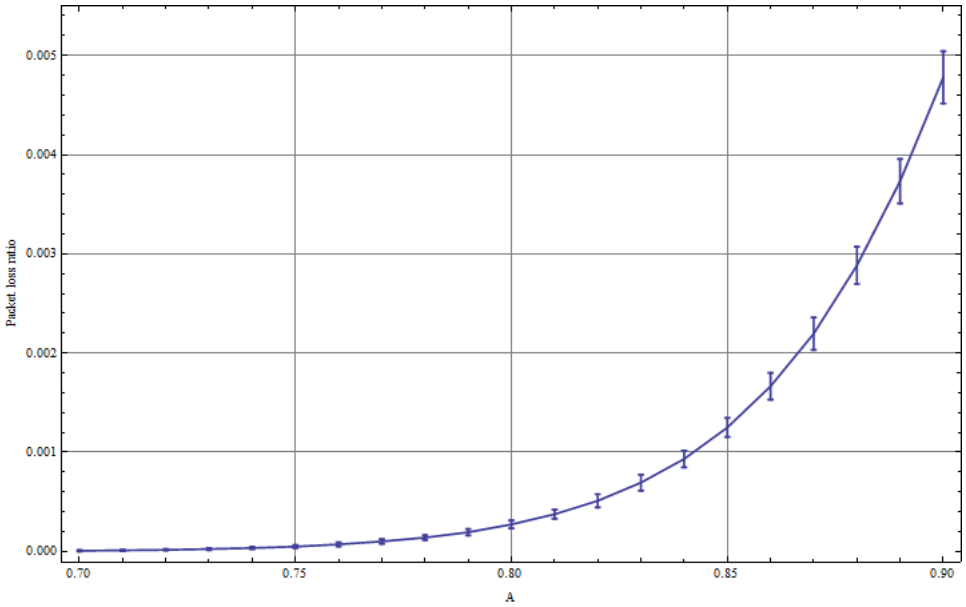

Figure 7.4: Average packet loss with increasing traffic load - IP simulator, Hyperexponential interarrival distribution

The average packet loss ratio is higher with the hyperexponential interarrival distribution compared to the N.E.D. interarrival distribution. The packet loss experienced between 0.7 and 0.75 Erlang is also low with this packet arrival process, but higher than with the N.E.D. interarrival distribution. The packet loss ratio is 6.82E-6 at 0.7 Erlang (16.7 dropped packets of 2444000 generated packets), and 4.93E-5 at 0.75 Erlang (129.3 dropped packets of 2619000 generated packets). A heavy increase in packet loss is observed above 0.8 Erlang.

The average packet loss ratio is higher with the hyperexponential interarrival distribution because of packet bursts arriving at output buffer 1 at some time instants. This increases the average filling of the output buffers and thus the average packet loss ratio.

7.2 OTN simulator

Simulation parameters used in the OTN simulator is seen from Table 7.2. These are the same parameters as used with the IP simulator. Each of the two output buffers have a buffer size of 12500 bytes.

Parameter	**Value**
Holdtime	10000
Transient simtime	2000
Number of replications	10
Simseed	30
Buffer size (bytes)	12500

Table 7.2: Simulation parameters, OTN simulator

The total buffer capacity in the OTN simulator is thus twice as big as in the IP simulator. This is because the two output buffers together share the total 100 Gb/s link capacity.

7.2.1 N.E.D. interarrival distribution

The OTN simulator is also first run with the negative exponential interarrival distribution.

Delay Figure 7.5 presents the average delay experienced per packet with increasing network load.

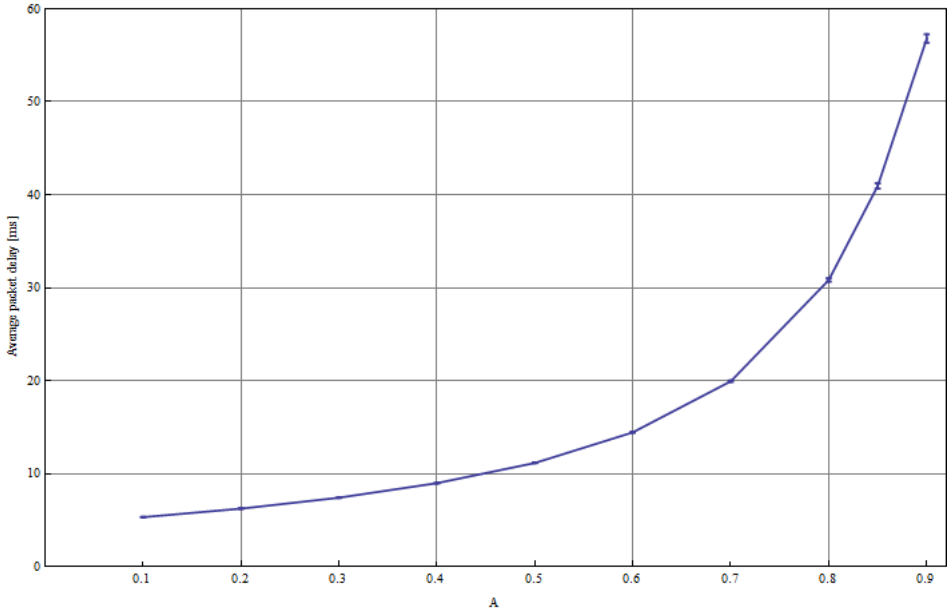

Figure 7.5: Average packet delay with increasing traffic load - OTN simulator, N.E.D. interarrival distribution

As seen from the figure; the average packet delay increases with increasing link load. This is, as with the IP simulator, because of the increased filling of the output buffers when the network load increases. Higher delay is experienced per packet with the OTN simulator than with the IP simulator. The average packet delay is 5.312 ms at 0.1 Erlang, and 56.81 ms at 0.9 Erlang.

Packet loss Figure 7.6 presents the average packet loss ratio (generated packets/average dropped packets) with increasing network load.

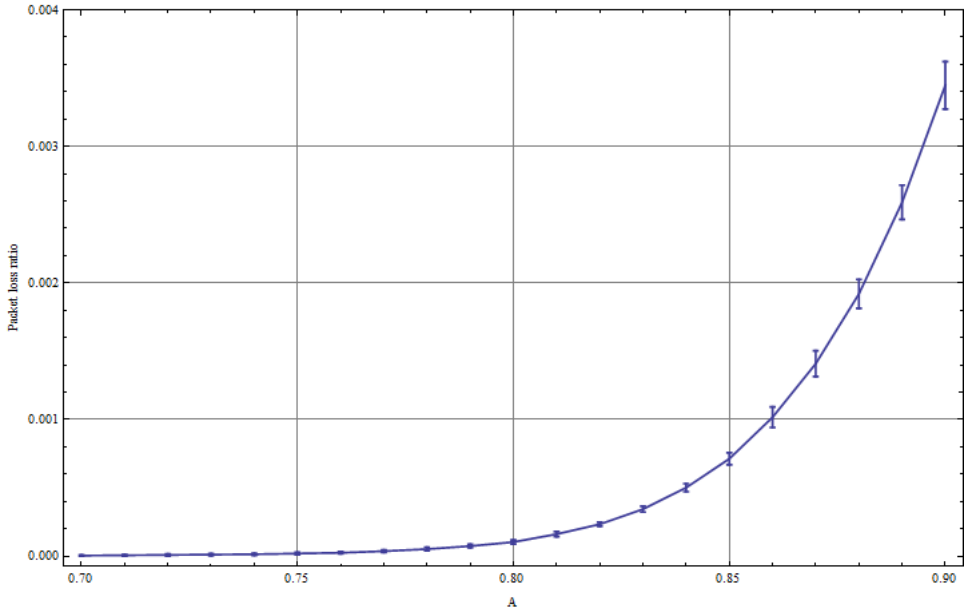

Figure 7.6: Average packet loss with increasing traffic load - OTN simulator, N.E.D. interarrival distribution

As seen from the graph; packet loss starts occuring around 0.7 Erlang and increases gradually above 0.75 Erlang. As experienced with the previous simulation runs; the packet loss is very low between 0.7 and 0.75 Erlang. A packet loss ratio of 3.85E-6 (9.4 dropped packets out of 2444000 generated packets) is experienced at 0.7 Erlang, and 1.88E-5 (49.2 dropped packets out of 2619000 generated packets) at 0.75 Erlang.

A heavy increase in packet loss ratio is observed above 0.8 Erlang.

7.2.2 Hyperexponential interarrival distribution

The hyperexponential interarrival distribution is now used to control the packet arrivals. The same burstiness parameters as with the IP simulator is used ($\lambda 2 = 50\ \lambda 1$, A = 0.2).

Delay Figure 7.7 presents the average packet delay with hyperexponential interarrival distribution.

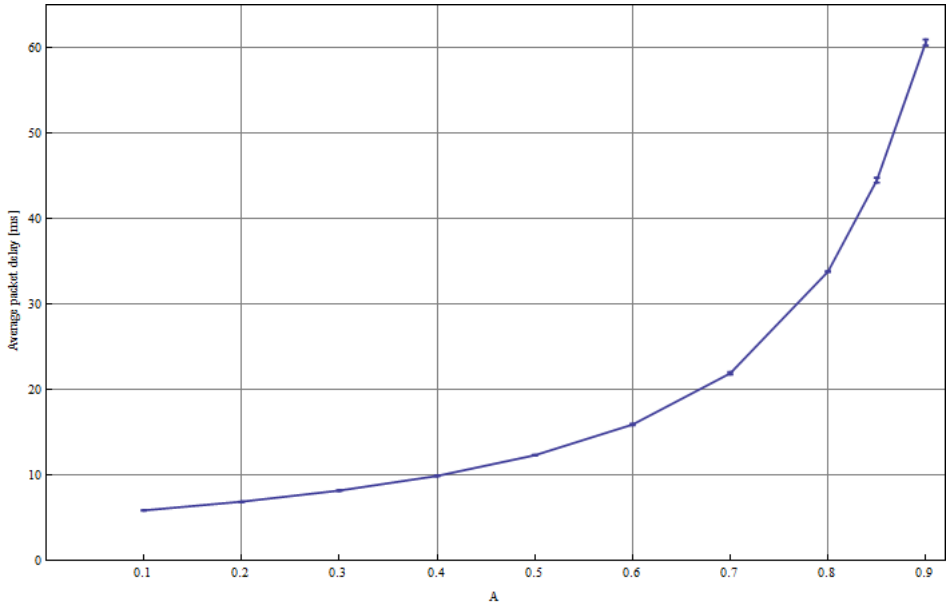

Figure 7.7: Average packet delay with increasing traffic load - OTN simulator, Hyperexponential interarrival distribution

The average packet delay is higher with the hyperexponential interarrival distribution than with N.E.D. interarrivals. This is, as with the IP simulator, because of the bursty traffic pattern that increases the filling of the buffers. When the filling of the output buffers on average is high, the average packet delay increases. The average packet delay is 5.85 ms at 0.1 Erlang, and 60.6 ms at 0.9 Erlang.

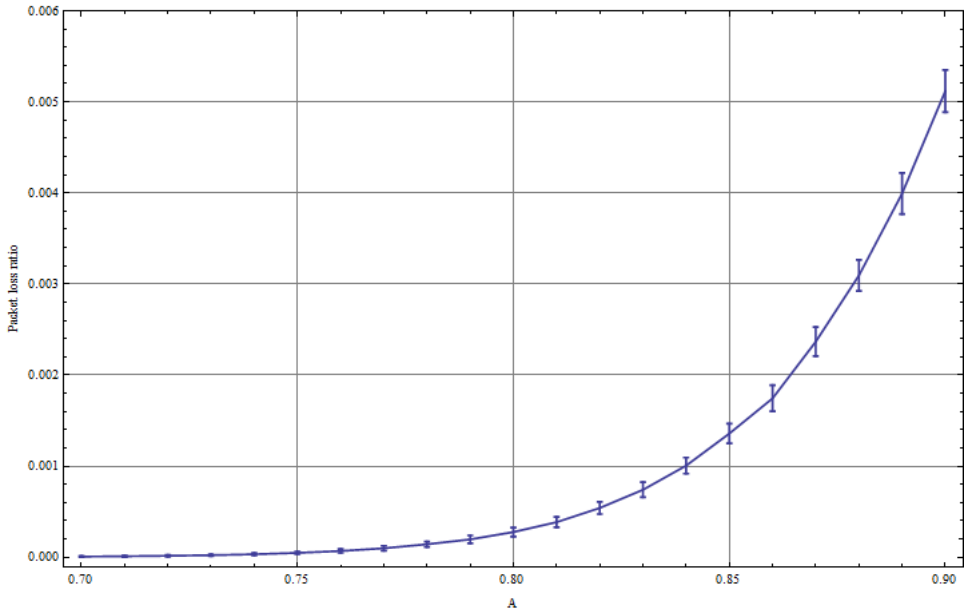

Figure 7.8: Average packet loss with increasing traffic load - OTN simulator, Hyperexponential interarrival distribution

Packet loss Figure 7.8 presents the average packet loss rate with hyperexponential interarrival distribution.

The packet loss rate is also higher for the OTN simulator with hyperexponential interarrival distribution than with N.E.D. interarrivals. The packet loss rate is still low in the area ranging from 0.7 to 0.75 Erlang; 6.99E-6 (17.1 dropped packets out of 2444000 generated packets) at 0.7 Erlang, and 4.79E-5 (125.6 dropped packets out of 2619000 generated packets) at 0.75 Erlang. The packet loss ratio increases heavily above 0.8 Erlang.

7.3 IP vs. OTN comparison

7.3.1 N.E.D. interarrival distribution

Delay A comparison of average packet delay experienced with the IP and OTN simulator with N.E.D. interarrival distribution is seen in Figure 7.9.

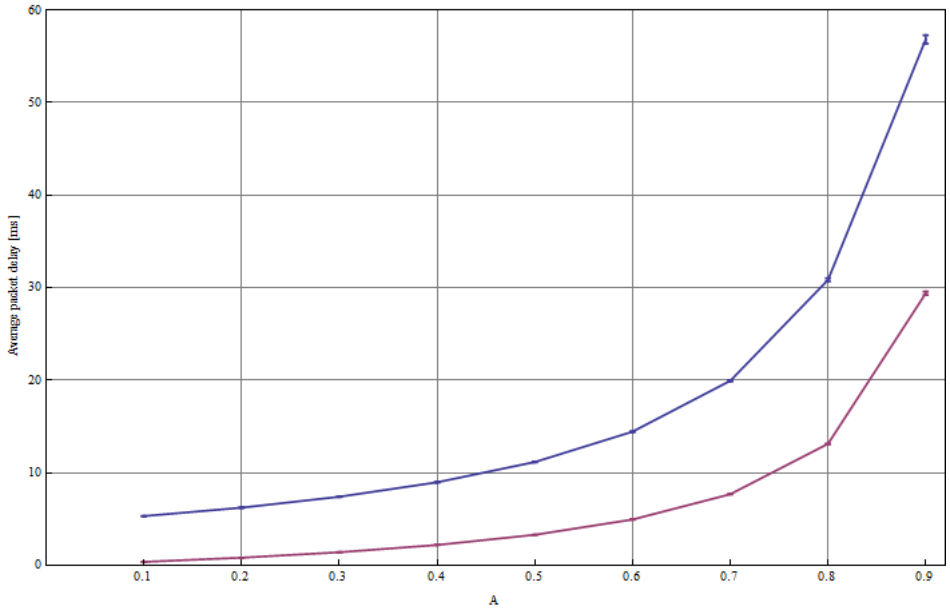

Figure 7.9: IP vs. OTN average packet delay comparison, N.E.D. interarrival distribution (Blue line (top) - OTN, purple line (bottom) - IP)

As seen from the graphs; the average packet delay experienced with OTN is higher than with IP. This is because the total transmission capacity is divided into smaller entities (sub-capacities) with OTN. 30% of the total link capacity (3 x 10 Gb/s) is statically configured to handle traffic destined to router 2, while 70% (7 x 10 Gb/s) is statically configured to handle traffic destined to router 3.

The total transmission capacity of 100 Gb/s is however shared between all packets on the links in the IP simulator. The transmission resources are thus higher utilized, improving the network performance. This is reffered to as the statistical multiplexing gain, which is achieved by using pure packet switching.

The average packet delay is thus lower for the IP simulator than the OTN simulator.

Packet loss A comparison of average packet loss with the IP and OTN simulator with N.E.D. interarrival distribution is seen in Figure 7.10.

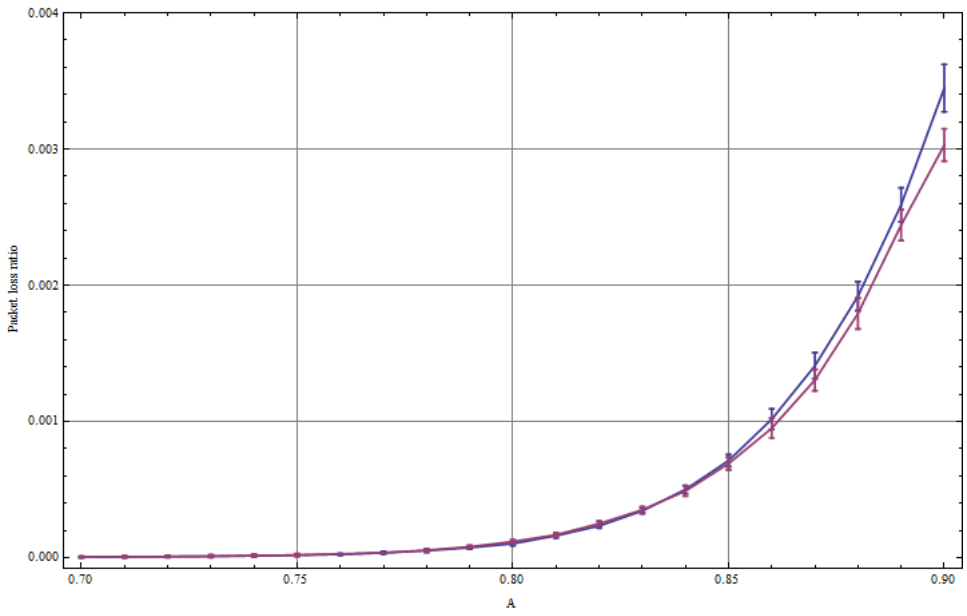

Figure 7.10: IP vs. OTN average packet loss comparison, N.E.D. interarrival distribution (Blue line (top) - OTN, purple line - IP (bottom))

As seen from Figure 7.10; the average packet loss rate experienced with both IP and OTN are overlapping up to about 0.85 Erlang. The packet loss rate increases more heavily for OTN than IP above 0.85 Erlang however. This is because of the statical assignment of transmission capacity to each output queue in the OTN scenario. Although the capacity is provisioned on basis of the expected traffic pattern (on average 30% of the traffic destined for router 2, 70% of the traffic destined for router 3), there will be variations in the actual traffic load. The problem arises when the distribution of traffic amongst the two output queues change compared to the expected traffic pattern. At some time instants, more than 30% of the generated traffic will be destined to router 2, or more than 70% to router 3. With pure packet switching this problem is avoided. Statistical multiplexing is performed in this case, and the total transmission capacity is fully shared amongst all packets regardless of destination. When the traffic pattern changes in comparison to what is expected with OTN however, a percentage of the total available bandwidth remains unused. One group of sub-lambdas might be fully congested while the other might be underutilized. A higher drop ratio is thus experienced

with OTN switching than with packet switching when the traffic load is high.

7.3.2 Hyperexponential interarrival distribution

Delay Figure 7.11 shows the average packet delay for IP vs. OTN with hyperexponential interarrival distribution.

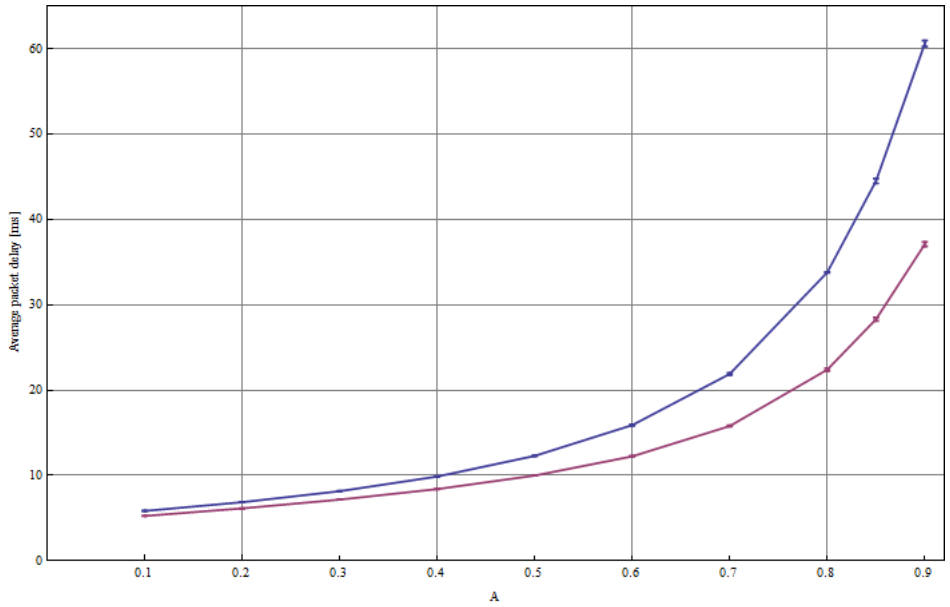

Figure 7.11: IP vs. OTN average packet delay comparison, Hyperexponential interarrival distribution (Blue line (top) - OTN, purple line (bottom) - IP)

The average packet delay is higher for both IP and OTN when the hyperexponential interarrival distribution is used. This is, as previously explained, because of random packet bursts that occur at some time instants. This causes the buffers to fill up with packets, thus increasing the average packet delay. As seen from the figure; average packet delay is about the same for both IP and OTN in the area ranging from 0.1 to 0.4 Erlang. The average packet delay experienced with OTN increases more heavily when the traffic load increases beyond 0.4 Erlang. This is, as with the N.E.D. interarrival distribution, experienced because of the static link configurations in OTN.

Packet loss Figure 7.12 shows the average packet loss rate for IP vs. OTN with hyperexponential interarrival distribution.

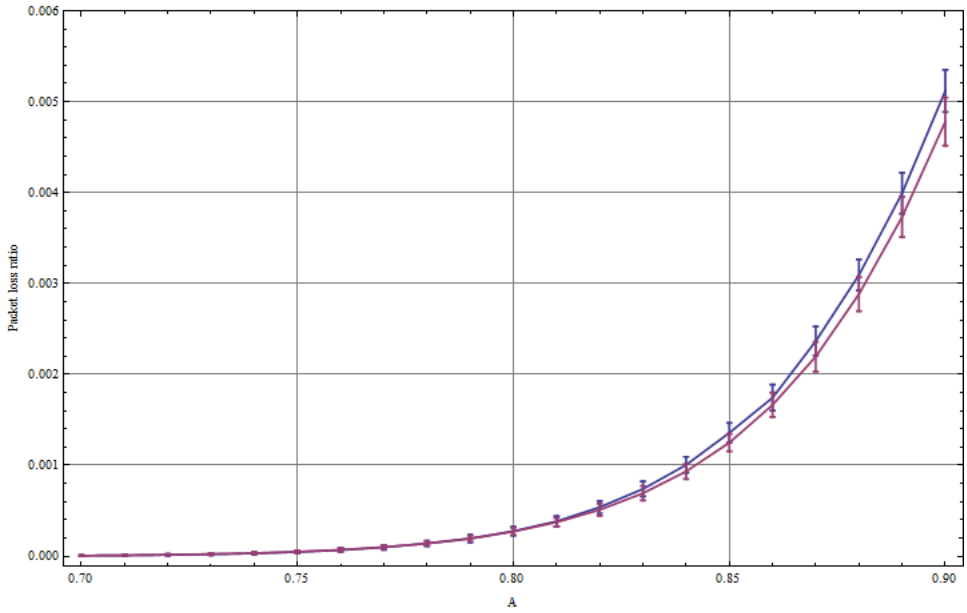

Figure 7.12: IP vs. OTN average packet loss comparison, Hyperexponential interarrival distribution (Blue line (top) - OTN, purple line (bottom) - IP)

As seen from the graphs; the packet loss rate is higher for both IP and OTN with the hyperexponential interarrival distribution compared to the N.E.D. interarrival distribution. The packet loss rate is similar for both IP and OTN up to 0.81 Erlang. The IP simulator is more tolerant to the bursty traffic pattern than the OTN simulator when the average traffic load increases above 0.81 Erlang. OTN has a higher average packet loss ratio than IP above this point. This is, as previously explained, because of the statically configured link capacities in OTN.

7.4 OTN Hyperexponential vs. N.E.D. interarrival distribution comparison

Figure 7.13 shows a comparison of average packet delay for the OTN simulator with hyperexponential and N.E.D. interarrival distributions.

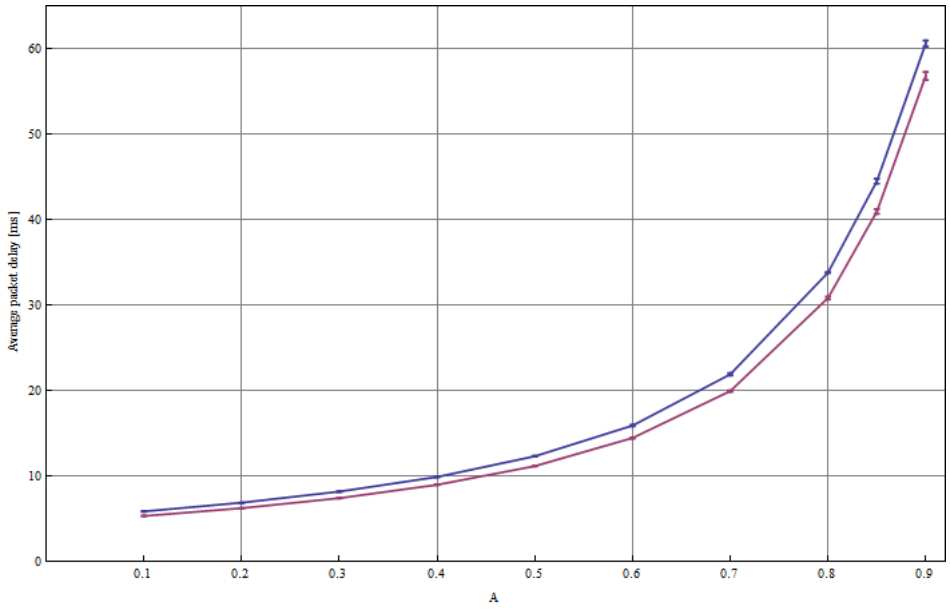

Figure 7.13: OTN average packet delay comparison with hyperexponential interarrival distribution vs. N.E.D. interarrival distribution, (Blue line (top) - Hyperexponential, purple line (bottom) - N.E.D.)

As seen from the graph; the average packet delay is higher with the hyperexponential than the N.E.D. interarrival distribution. The higher average packet delay is, as previously explained, because of traffic bursts that are generated with the hyperexponential distribution. The difference in average packet delay increases with increasing average network load. The difference is however not very big. The biggest difference in average packet delay is found at 0.9 Erlang, where the difference is 3.79 ms.

Figure 7.14 shows a comparison of average packet loss for the OTN simulator with hyperexponential and N.E.D. interarrival distributions.

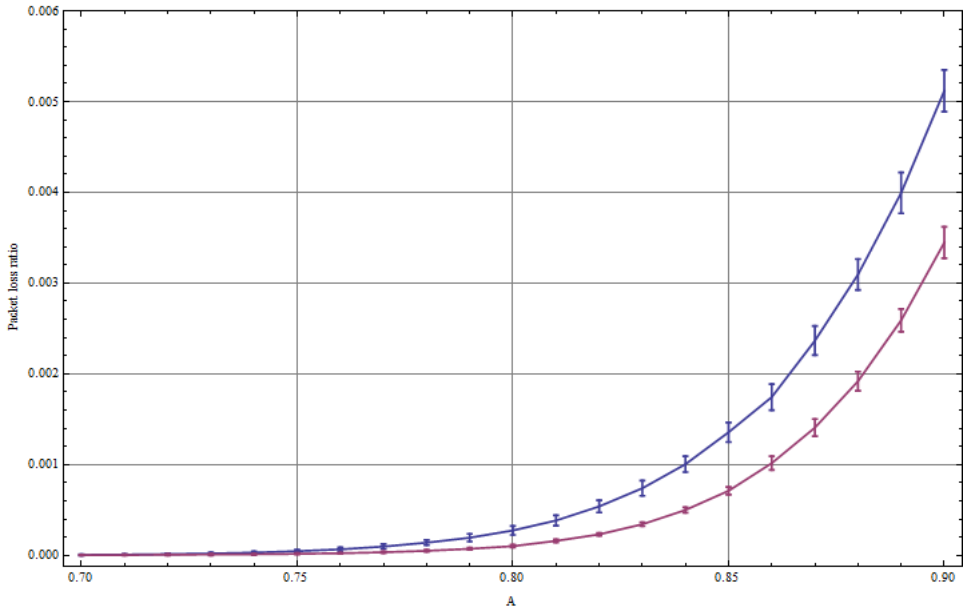

Figure 7.14: OTN average packet loss comparison with hyperexponential interarrival distribution vs. N.E.D. interarrival distribution, (Blue line (top) - Hyperexponential, purple line (bottom) - N.E.D.)

As seen from the graphs; the average packet loss ratio is very low from 0.7 to 0.75 Erlang with both distributions. The difference in packet loss ratio is however clearly seen above 0.75 Erlang. From this point, the packet loss ratio with the hyperexponential interarrival distribution increases much more heavily than with the N.E.D. interarrival distribution, and the gap increases with increasing traffic load. It is clearly seen from the hyperexponential loss graph that the statically configured OTN switched system doesn't handle the bursty traffic pattern very well.

7.5 Reconfigurable OTN simulator

The reconfigurable OTN switch is, as previously explained, capable of reconfiguring link sub-capacities based on the current traffic pattern. The queue length of the two output buffers are measured and compared for each packet that is generated. The capacity of the link connected to the output buffer with more packets is increased with 1 Gb/s. No adjustment of the link capacities occur if the output queues have the same queue length, or if the limits of capacity rearrangement is reached.

The adjustment of capacity is limited so that the capacity of the link connected to R2 can vary between 25 and 50 Gb/s, while the link connected to R3 can vary between 50 and 75 Gb/s.

Simulation parameters used in the reconfigurable OTN simulator is seen from Table 7.3. These are the same parameters as used with the IP and OTN simulators. Each output buffer has a capacity of 12500 bytes.

Parameter	**Value**
Holdtime	10000
Transient simtime	2000
Number of replications	10
Simseed	30
Buffer size (bytes)	12500

Table 7.3: Simulation parameters, Reconfigurable OTN simulator

7.5.1 N.E.D. interarrival distribution

The reconfigurable OTN simulator is first run with the N.E.D. interarrival distribution.

Delay Figure 7.15 presents the average packet delay experienced with increasing link load.

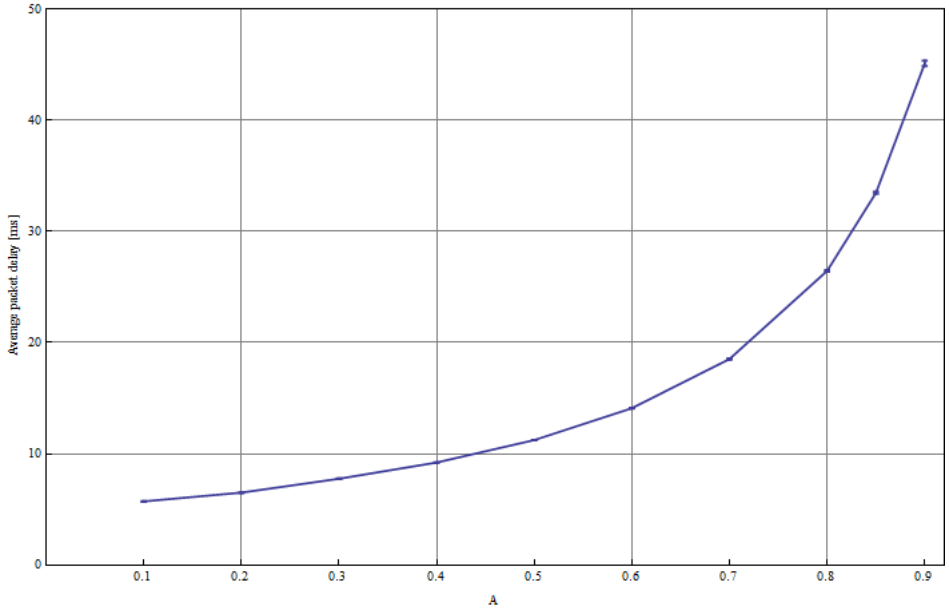

Figure 7.15: Average packet delay with increasing traffic load - Reconfigurable OTN simulator, N.E.D. interarrival distribution

As seen from the graph; the packet delay increases when the average link load increases. The average packet delay is 5.573 ms at 0.1 Erlang and 45.1 ms at 0.9 Erlang.

Loss Figure 7.16 presents the average packet loss with increasing link load.

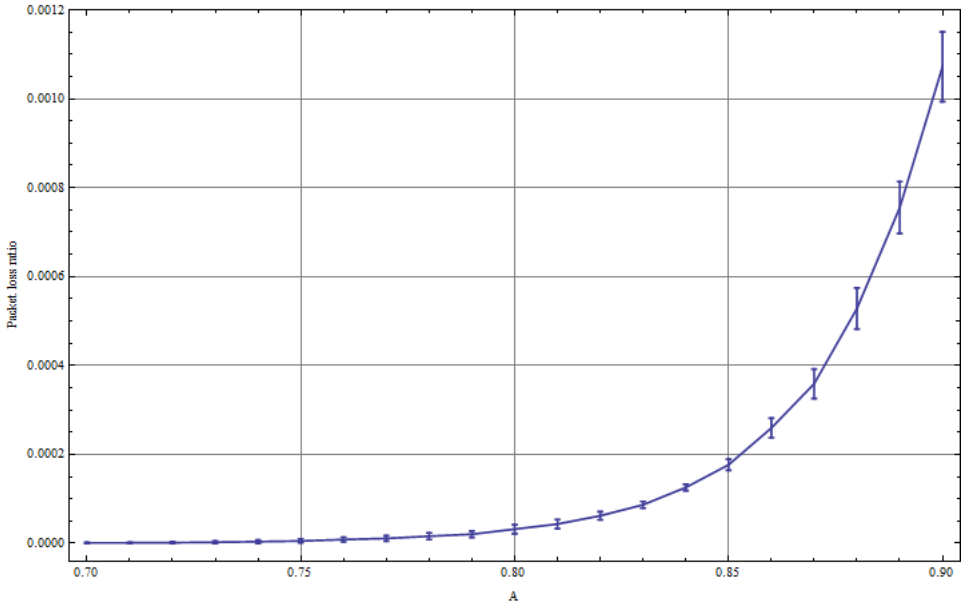

Figure 7.16: Average packet loss with increasing traffic load - Reconfigurable OTN simulator, N.E.D. interarrival distribution

As seen from the graph; the packet loss is very low from 0.7 to 0.8 Erlang. The packet loss is 9.82E-7 (2.4 dropped packets out of 2444000 generated packets) at 0.7 Erlang, and 3.33E-5 (93.1 dropped packets out of 2793000 generated packets) at 0.8 Erlang.

7.5.2 Hyperexponential interarrival distribution

The reconfigurable OTN simulator is now run with the hyperexponential interarrival distribution. The same burstiness parameters as with the IP and regular OTN simulators are used ($\lambda 2 = 50\ \lambda 1$, A = 0.2).

Delay Figure 7.17 presents the average packet delay experienced with increasing link load.

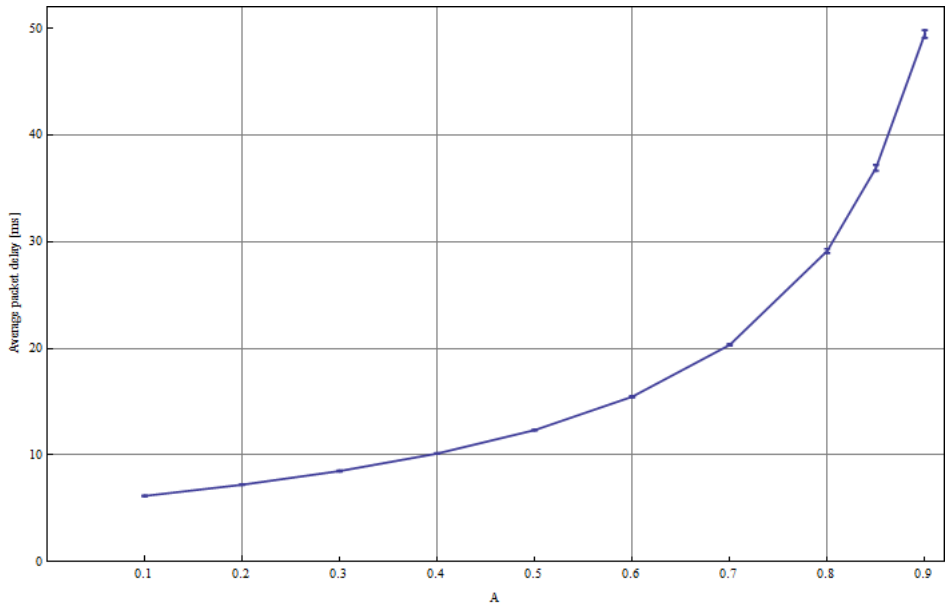

Figure 7.17: Average packet delay with increasing traffic load - Reconfigurable OTN simulator, Hyperexponential interarrival distribution

The average packet delay is higher with the hyperexponential interarrival distribution than with the N.E.D. interarrival distribution. The packet delay is 6.18 ms at 0.1 Erlang, and 49.5 ms at 0.9 Erlang.

Loss Figure 7.18 presents the average packet loss with increasing link load.

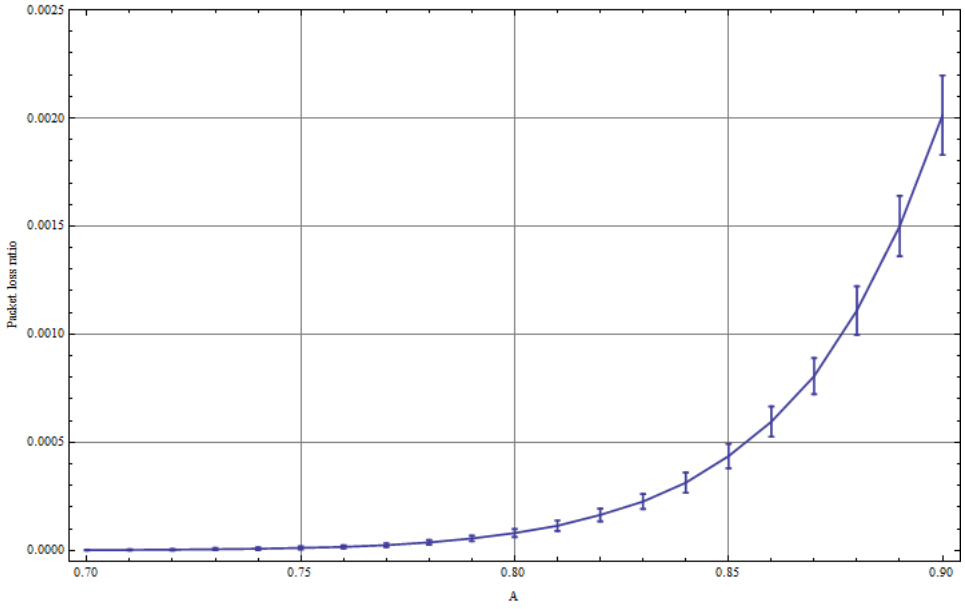

Figure 7.18: Average packet loss with increasing traffic load - Reconfigurable OTN simulator, Hyperexponential interarrival distribution

As seen from the graph; the packet loss is very low from 0.7 to 0.75 Erlang with the hyperexponential interarrival distribution. The packet loss is 2.046E-6 (5 dropped packets out of 2444000 generated packets) at 0.7 Erlang, and 1.214E-5 (31.8 dropped packets out of 2619000 generated packets) at 0.75 Erlang. The packet loss rate increases gradually above 0.75 Erlang.

7.6 IP vs. OTN vs. reconfigurable OTN comparison

7.6.1 N.E.D. interarrival distribution

Delay Figure 7.19 presents the average packet delay with increasing link load for the IP, OTN and reconfigurable OTN simulator with the N.E.D. interarrival distribution.

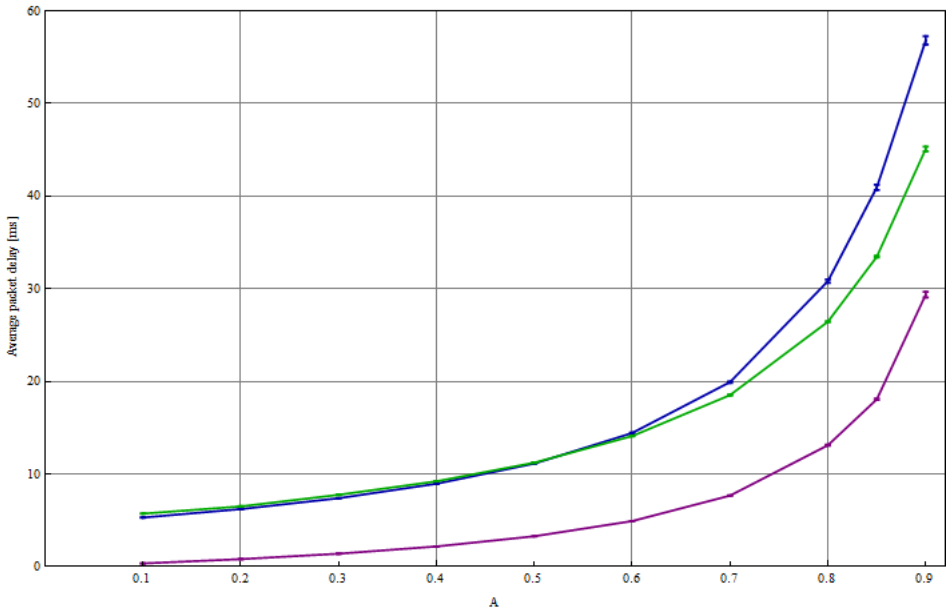

Figure 7.19: Average packet delay with increasing traffic load - IP vs. OTN vs. reconfigurable OTN simulator, N.E.D. interarrival distribution, (Blue line (top) - OTN, green line (middle) - reconfigurable OTN, purple line (bottom) - IP)

As seen from the graphs; the average packet delay for OTN and reconfigurable OTN is similar up to 0.6 Erlang. The packet delay increases more heavily for OTN than reconfigurable OTN above 0.6 Erlang, however. This is because reconfiguration of link capacities in the reconfigurable OTN simulator starts occuring above 0.6 Erlang, while the regular OTN simulator has static link capacities. The reconfigurable OTN simulator makes sure that the output buffer with highest queue length gets its link capacity increased. The average time spent in the output buffers for packets in the reconfigurable OTN simulator is thus lower than with the regular OTN simulator, and the

average packet delay is thus lower for the reconfigurable OTN simulator.

The average packet delay for the IP simulator is however lower than both the regular and reconfigurable OTN simulator. This is, as previously explained, because of the statistical multiplexing gain in the IP simulator, where the total transmission resource is shared between all packets.

Loss Figure 7.20 presents the average packet loss with increasing link load for the IP, OTN and reconfigurable OTN simulator with the N.E.D. interarrival distribution.

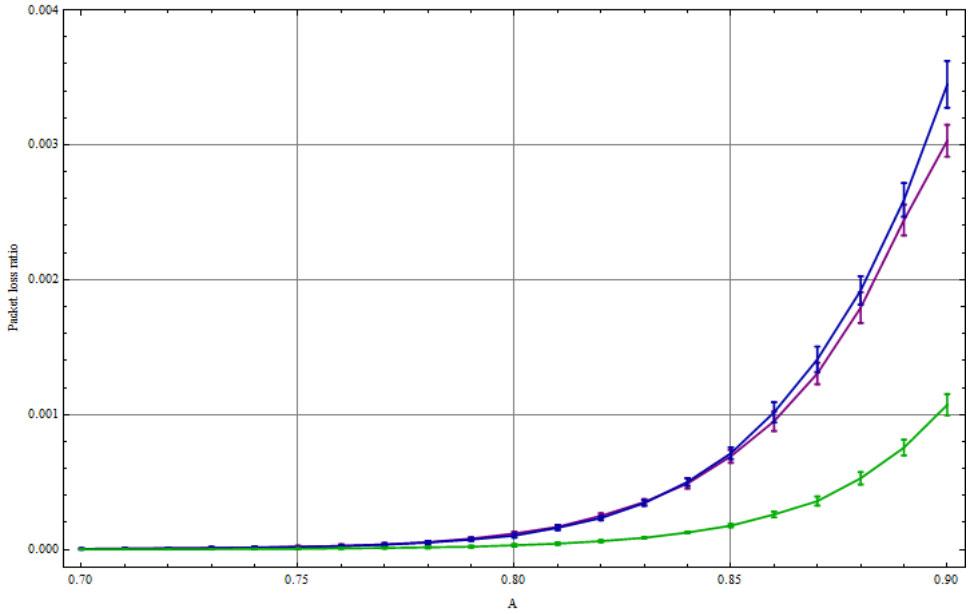

Figure 7.20: Average packet loss with increasing traffic load - IP vs. OTN vs. reconfigurable OTN simulator, N.E.D. interarrival distribution, (Blue line (top) - OTN, purple line (middle) - IP, green line (bottom) - reconfigurable OTN)

As seen from the graphs; the average packet loss experienced with the reconfigurable OTN simulator is much lower than with the regular OTN and IP simulators. The difference in packet loss is seen from 0.76 Erlang and above, where the packet loss of the IP and OTN simulators increases more heavily than the reconfigurable OTN simulator.

The packet loss ratio of both IP and regular OTN increases heavily above 0.8 Erlang. Both have the same increase in packet loss ratio up to 0.85 Erlang. The OTN simulator has a higher packet loss ratio than IP above 0.85 Erlang.

The increase in packet loss ratio with increasing traffic load for the reconfigurable OTN simulator is much lower than with the OTN and IP simulators. An increase in packet loss ratio is first barely seen above 0.8 Erlang with the reconfigurable OTN simulator, and the packet loss ratio has a much less steep curve than the IP and OTN simulators. At 0.9 Erlang, where the packet loss ratio is at its highest, the packet loss ratio is 0.00327 (on average 10279.6 dropped packets out of 3143000 generated packets) for the IP simulator and 0.003447 (on average 10834.5 dropped packets out of 3143000 generated packets for the OTN simulator. The reconfigurable OTN simulator has however a packet loss ratio of only 0.00107 (on average 3369.3 dropped packets out of 3143000 generated packets). The low packet loss ratio experienced with the reconfigurable OTN simulator is because of the dynamical adjustment of the link capacites. Measuring the potential difference in queue lengths for each packet that is generated makes the system responsive to changes in the traffic pattern. A link is assigned extra transmission capacity at once if the output queue connected to it starts growing in length. Assigning higher link capacity makes the queue length decrease rapidly. In this way, maximum filling of the buffers is avoided, and a lower packet loss rate is achieved.

7.6.2 Hyperexponential interarrival distribution

Figure 7.21 presents the average packet delay with increasing link load for the IP, OTN and reconfigurable OTN simulator with the hyperexponential interarrival distribution.

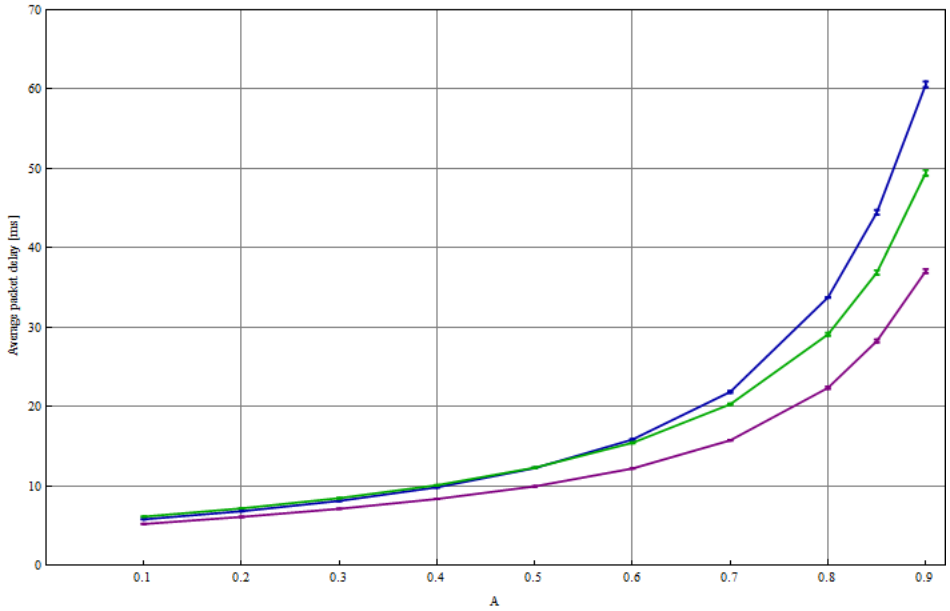

Figure 7.21: Average packet delay with increasing traffic load - IP vs. OTN vs. reconfigurable OTN simulator, Hyperexponential interarrival distribution, (Blue line (top) - OTN, green line (middle) - reconfigurable OTN, purple line (bottom) - IP)

Delay Still, the average packet delay of the IP simulator is lowest and the average packet delay of OTN and reconfigurable OTN is overlapping up to 0.6 Erlang. The average packet delay of OTN increases more than reconfigurable OTN above 0.6 Erlang. This is, as previously explained, because the reconfiguration of link capacities in the reconfigurable OTN simulator starts occuring above this point. The reconfiguration of link capacities makes sure that the buffer with most packets gets its link capacity increased, thus decreasing the overall average packet delay. The gap in packet delay between the IP simulator and the OTN simulators is however smaller than with the N.E.D. interarrival distribution. This is because the buffers on average contain more packets with the hyperexponential interarrival distribution.

Loss Figure 7.22 presents the average packet loss with increasing link load for the IP, OTN and reconfigurable OTN simulator with the hyperexponential interarrival distribution.

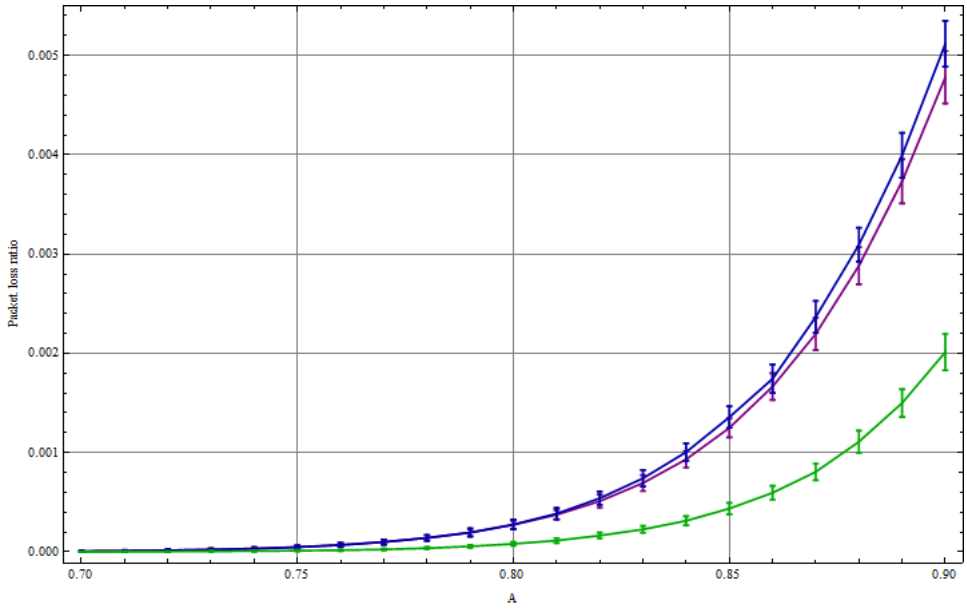

Figure 7.22: Average packet loss with increasing traffic load - IP vs. OTN vs. reconfigurable OTN simulator, Hyperexponential interarrival distribution, (Blue line (top) - OTN, purple line (middle) - IP, green line (bottom) - reconfigurable OTN)

A heavier increase in packet loss rate with increasing average traffic load than with the N.E.D. interarrival distribution is observed for all simulators. This is because of the bursty traffic pattern generated by the hyperexponential interarrival distribution. The packet loss ratio of IP and regular OTN is overlapping up to 0.81 Erlang, while the packet loss ratio with reconfigurable OTN is much lower. At 0.9 Erlang, the packet loss ratio is 0.00478 (on average 15045.1 dropped packets out of 3143000 generated packets) for the IP simulator, and 0.00512 (on average 16108.4 dropped packets out of 3143000 generated packets for the OTN simulator). The reconfigurable OTN simulator has however a packet loss ratio of only 0.00201 (on average 6334.3 dropped packets out of 3143000 generated packets).

The difference in packet loss ratio between regular OTN and IP, and reconfigurable OTN is seen already from 0.75 Erlang with the hyperexponential interarrival distribution.

7.7 Discussion

The results are better both in terms of average packet delay (lower delay) and packet loss (lower loss) for the IP simulator than the regular OTN simulator, both with N.E.D. and hyperexponential interarrival distribution.

The results illustrates the performance benefit of sharing transmission resources and performing statistical multiplexing (IP), rather than statically assigning transmission capacity based on expected traffic loads using time division multiplexing (OTN). The main advantage of packet switching is, as seen from the simulation results, the statistical multiplexing gain. The transmission resources are fully shared, and the link sharing is instantaneously adapted to the traffic demands of the data streams. High link utilization and tolerance to sudden changes in the traffic pattern is thus achieved. Regular OTN is, on the other hand, statically configured based on expected link loads. The link utilization is of this reason lower, and the system isn't tolerant to changes in the traffic pattern.

The reconfigurable OTN simulator provides however better results than both the IP simulator (lower packet loss) and the regular OTN simulator (lower delay above 0.6 Erlang and lower packet loss). This is because the reconfigurable OTN simulator combines two important features; bypass of the intermediate IP router, and dynamic reconfiguration of link capacities. In this way, both output links are fully utilized, and the system automatically adapts to changes in the traffic pattern. It thus combines the benefits of packet switching and circuit switching. The reconfigurable OTN simulator did however perform queue length checking for each packet that was generated. This is not practically feasible in real-life switches. A real-life OTN switch will have limitations in the electronics that put constraints on how often the checking of output buffer queue lengths can be performed. A real-life implementation of the reconfigurable OTN switch will thus have poorer performance than the one that is proposed in this thesis.

The main benefit of the regular OTN switched network approach is the predictable behavior of the circuit switched configuration. The bandwidth is fixed, making it possible to give guarantees for bandwidth, delay and packet loss for separate connections. This is different from the unpredictable nature of packet switching where all traffic shares the same bandwidth. Circuit switching involves provisioning static connections. This works fine as long as the traffic pattern keeps somewhat the same. As experienced from the simulation runs; the problem arises when the traffic pattern change. Because the connections with given bandwidths are nailed up, there is no dynamic rearrangement of the available bandwidth. Some connections might in worst case turn idle, being totally unused, while other circuits are overloaded, resulting

in loss and increased delays. In presence of variation in traffic pattern, bursts and flow demands, circuit switched networks have to conservatively allocate bandwidth to handle peak capacity of the flow. The utilization of link capacities is thus lower than with packet switching, which shares the transmission capacity, and performs statistical multiplexing based on current bandwidth demands from the various flows.

Employing packet switching with statistical multiplexing in the network does however mean that a "bet" is placed, hoping that multiple flows will make it through the the pipe that is provisioned. If the required bandwidth for the sum of peaks of all flows exceeds the bandwidth that is provisioned, congestion, and subsequently packet loss will occur.

Using OTN switches in the core network provides a potential simpler network design. The OTN switches work as a transport layer, switching aggregated data at coarse granularities between nodes in the core network. Unnecessary processing of transit traffic in intermediate IP routers is avoided, reducing the requirement for processing power, line card interface bit rates and number of ports in the routers. Implementation of the core network is thus potentially simpler and more cost-effective if OTN switches are used as a transport layer beneath the IP routers.

OTN switching also offers lower equipment cost than IP routing (OTN switches are cheaper than IP routers), but has, as seen from the simulation results, poorer performance. The performance cost of using OTN switches with circuit switching instead of IP routers with pure packet switching is seen both in higher packet delay and loss. The average packet delay is higher with OTN than with IP at all traffic loads. This is, as previously explained, because of the division into link sub-capacities with OTN. The average packet loss is however similar for both OTN and IP up to 0.85 Erlang (N.E.D. interarrival), and 0.81 Erlang (Hyperexponential interarrival).

Chapter 8

Conclusion

The choice of an adequate core switching architecture is crucial in order to handle the heavy traffic growth in the Internet. Switching technology used in core network nodes shouldn't be the transmission bottleneck, but rather make sure that the fiber capacity is highly utilized, and perform data forwarding in the most efficient way possible.

Optical packet switching is under heavy research, and is the future vision of core network evolution. Many obstacles are still to overcome before this technology will be seen in real-life.

OTN is gradually replacing SDH-based transport networks. OTN is based on a range of ITU-T standards, and takes single wavelength SDH technology a step further, providing future-proof multi-wavelength manageable transport networks. Protocol transparency makes OTN the transport vehicle for any type of service, and the TDM multiplexing hierarchy provides high utilization of wavelength capacity. With OTN, the operators can switch payloads that contain SDH, Ethernet and IP within a unified transport layer. OTN switching makes it possible to switch circuit services at the circuit layer (lambda or OTN), and to switch packet services at the packet layer (layer 3). This makes it possible for operators to perform data transport and switching in the most cost-effective layer possible.

Data is organized in ODUs when OTN is used as transport technology. The ODU is used as the basic multiplexing unit in the TDM hierarchy, and can contain any type of supported client signal. An OTN switch capable of switching ODUs has been proposed in this thesis. OTN switching was then compared to using pure packet switching in a three-node network scenario.

The simulation results show better performance for packet switching than regular OTN switching. Both average packet delay and packet loss is found to be lower with the IP simulator than the regular OTN simulator. This is because of the static bandwidth assignment that is performed with regular

OTN switching. Nailed-up link capacities are configured based on expected traffic patterns. Time-division multiplexing used in OTN switches doesn't provide any performance benefits compared to statistical multiplexing performed with IP routers. The performance benefit gained with choosing a regular OTN switching layer beneath the IP routing layer is bypass of intermediate IP routers, thus saving the intermediate IP routers for unnecessary packet processing. OTN provides a potential network design benefit because of its ability to effectively switch aggregated traffic at coarse granularities in the OTN layer, and forward traffic destined to IP routers in network nodes for fine-granularity routing. The network design benefit does however come at the cost of poorer network performance because of the time division multiplexing performed by OTN switches.

The financial expenses of OTN switches is lower than IP routers. A cost-effective core network architecture is thus possible to achieve if OTN switches is used in combination with IP routers. Combining an OTN switch with the IP router in each network node makes it possible to reduce the required IP router processing capacity in each node, thus reducing the overall network cost.

The reconfigurable OTN switch provides better performance results than both regular OTN switching, and packet switching. The combination of both circuit and packet switching clearly has its benefits. The reconfigurable OTN switch proposed in this thesis did however perform reconfiguration of link capacities by comparing the output queue lengths for each packet that is handled. Checking of output buffer queue lengths with this interval is difficult to achieve in real-life switches.

Many considerations must be made when the core network switching architecture is chosen. OTN switching provides a cost-effective alternative to a pure packet switched core although the performance of regular OTN switching isn't as good as packet switching. Reconfigurable OTN switching combines the features of the regular OTN switch with the features of packet switching, providing efficient network node bypass, and high link utilization. Reconfigurable OTN switching might thus be the future optimum solution for core network switching.

8.1 Further work

Performance evaluation of OTN switching vs. packet switching took place in a three-node network scenario in this thesis. Further work may involve extending the evaluation by testing OTN switching in a larger scale network scenario.

References

[1] Michael S. Borella et al. Optical components for wdm lightwave networks. *Proceedings of the IEEE, Vol. 85, No. 8, pages 1274-1307*, August 1997.

[2] Steinar Bjørnstad. Terabit kapasitets nettverk ved bruk av optikk. Telenor FoU- Fornebu/NTNU - Trondheim/UNIK - Kjeller.

[3] E. Zouganeli. Optical networks: From point-to-point transmission to full networking capabilities. *Telektronikk*, February 2005.

[4] International Telecommunication Union. Itu-t g.709, interfaces for the optical transport network. Recommendation, 2009.

[5] International Telecommunication Union. Optical transport network tutorial, 2005. URL: https://www.itu.int/ITU-T/2005-2008/com15/otn/OTNtutorial.pdf.

[6] Jan A. Audestad. *Technologies and systems for access and transport networks*. Artech House, 2008.

[7] Kevin Thompson K. Claffy, Greg Miller. The nature of the beast: Recent traffic measurements from an internet backbone, 1998. URL: http://www.caida.org/publications/papers/1998/Inet98/Inet98.pdf.

[8] Cisco Systems. Cisco visual networking index. White paper, June 2010. URL: http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360.pdf.

[9] Manohar Naidu Ellanti Lakshmi G. Raman et al. *Next Generation Transport Networks - Data, Management and Control Planes*. Springer, 2005.

[10] Filip Idzikowski. Power consumption of network elements in ip over wdm networks. Technical report, Berlin Tecnical University, Telecommunication Networks Group, July 2009. URL: http://www.tkn.tu-berlin.de/publications/papers/powerNumbers_final.pdf.

[11] About itu. URL: http://www.itu.int/net/about/.

[12] Per Harald Knudsen-Baas. Otn switching. Technical report, NTNU, Department of Telematics, December 2010.

[13] International Telecommunication Union. Optical fibres, cables and systems. ITU-T Manual, 2009. URL: http://www.itu.int/dms_pub/itu-t/opb/hdb/T-HDB-OUT.10-2009-1-PDF-E.pdf.

[14] K. N. Sivarjan R. Ramaswami. *Optical Networks - a practical perspective*. Academic Press, 2002.

[15] Steve Gorshe. A tutorial on itu-t g.709 optical transport networks (otn). White paper, PMC-Sierra, 2010. URL: http://www.elettronicanews.it/01NET/Photo_Library/775/PMC_OTN_pdf.pdf.

[16] Osamu Ishida Takuya Ohara. Standardization activities for the optical transport network. *NTT Technical Review, Vol. 7 No. 3*, 2009.

[17] TPack. Odu0 and oduflex - a future-proof solution for otn client mapping. White paper, 2010.

[18] Andreas Schubert. G.709 - the optical transport network (otn). JDSU, White paper, 2007.

[19] Itu-t study group 15. URL: http://www.itu.int/ITU-T/studygroups/com15/index.asp.

[20] Martin Carroll et al. The operator's view of otn evolution. *Communications Magazine, IEEE*, pages 46–52, 2010.

[21] Ieee p802.3ba 40gb/s and 100gb/s ethernet task force. URL: http://www.ieee802.org/3/ba/.

[22] M.J. O'Mahony, D. Simeonidou, D.K. Hunter, and A. Tzanakaki. The application of optical packet switching in future communication networks. *Communications Magazine, IEEE*, 39(3):128–135, mar 2001.

[23] Matthias Berger et al. Optical transport networks (otn) - techincal trends and assessment. ITG, 2006. URL: http://www.eusar.de/NR/rdonlyres/6633CE02-7567-4143-A62A-99FC2E083C03/14356/ITGPosipapOTN1.pdf.

[24] Bjarne E. Helvik Peder J. Emstad, Poul E. Heegard and Laurent Paquereau. *TTM4110 Compendium, Dependability and performance in information and communication systems - Fundamentals*. Tapir akademisk forlag, 2008.

[25] H. Øverby N. Stol. Effects of bursty traffic in service differentiated optical packet switched networks. *OSA Optics Express 12(3), p. 410-415*, 2004.

[26] International Telecommunication Union. Itu-t g.872, architecture of optical transport networks. Recommendation, 2001.

Appendix A

Simulator source code

A.1 IP simulator

A.1.1 N.E.D. interarrival distribution

```
BEGIN

external class demos="C:\cim\demos\demos_new.atr";

demos begin

INTEGER bufferSize; comment Bytes;
INTEGER randomNumber;
INTEGER hopCount;
INTEGER i;
INTEGER linkCapacity;
INTEGER packetSum;
INTEGER simSeed,simTime,transient_simtime,numReps;
REAL packetLength;
REAL averageLoad,packetsPerTimeUnit,averageBitrate;
LONG REAL averagePacketLength;

REF(count) generatedPackets,droppedPackets,receivedPackets;
REF(rdist) packetLengthDist,interarrivalDist;
REF(tally) packetDelay;
REF(idist) r;
REF(waitq) r_out_q1,r_out_q2;
REF(infile) packetDist_infile;


COMMENT **********Classes**********;

COMMENT **********PACKET GENERATOR CLASS**********;

entity CLASS packetGenerator;
Begin

INTEGER i;
i:=1;
```

```
loop:

comment **********GENERATING PACKETS DESTINED TO R1**********;

        new packet(edit("packet",i),1).schedule(0.0);

IF time > transient_simtime THEN BEGIN

        generatedPackets.update(1);

        packetSum:=packetSum+1;

END;

        hold(interarrivalDist.sample);

        i:=i+1;

        repeat;

End;


COMMENT ********************PACKET CLASS**********************;
entity CLASS packet(hopCount);

INTEGER hopCount;

Begin

        REAL timestamp;
        REAL packetLength,maxSize;

        timestamp:=time;


COMMENT ********************PACKET SIZING**********************;

        maxSize := 1500; COMMENT max ip payload size;

        packetLength:= packetLengthDist.sample;

        IF packetLength>0 THEN
        BEGIN

         IF packetLength > maxSize THEN
         BEGIN

                packetLength := packetLength - maxSize;

         END;

        END;


COMMENT ****************end of packet sizing********************;


COMMENT **************PUTTING PACKET IN QUEUE******************;


IF (hopCount=1) THEN BEGIN
```

```
        IF r_out_q1.length*packetLengthDist.avg < bufferSize THEN BEGIN

                r_out_q1.wait;
                comment outtext("Putting packet in q1");
                comment outimage;

        END ELSE BEGIN

                IF time > transient_simtime THEN BEGIN

                droppedPackets.update(1);
                comment outtext("Dropping packet from q1");
                comment outimage;

        END;

        END;

END;

IF (hopCount=2) THEN BEGIN

        IF r_out_q2.length*packetLengthDist.avg < bufferSize THEN BEGIN

        r_out_q2.wait;
        comment outtext("Putting packet in q2");
        comment outimage;

        END ELSE BEGIN

        IF time > transient_simtime THEN BEGIN

                droppedPackets.update(1);
                comment outtext("Dropping packet from q2");
                comment outimage;

        END;

END;

END;

End;


comment ******************IP ROUTER CLASS************************;
entity CLASS ip_router(outQueue);

ref(waitq) outQueue;

Begin

ref(packet) packet_;

LOOP:

packet_:-outQueue.coopt;

IF time>0 THEN BEGIN

        hold(((packet_.packetLength)*8)/(linkCapacity));
```

```
        comment Transmission delay;

        comment outtext("Holding for: ");
        comment outfix(((packet_.packetLength)*8)/(linkCapacity),5,10);
        comment outimage;

END;

        packet_.hopCount:=packet_.hopCount+1;


comment *******CHECKING DESTINATION NODE --
-- 0,7 PROBABILITY OF TRANSIT IN ROUTER 2.
IF AT ROUTER 3 - NO TRANSIT********;

IF packet_.hopCount=2 THEN BEGIN

        randomNumber:=r.sample;

        IF randomNumber>3 THEN BEGIN

        packet_.schedule(now);

        comment outtext("Transiting packet to R3");
        comment outimage;

        END ELSE BEGIN

        IF time > transient_simtime THEN BEGIN

                packetDelay.update(time-(packet_.timestamp));

                comment outtext("Dropping packet at R2");
                comment outimage;

                receivedPackets.update(1);

        END;

        END;

END;

IF packet_.hopCount=3 THEN BEGIN

IF time > transient_simtime THEN BEGIN

        packetDelay.update(time-(packet_.timestamp));

        comment outtext("Dropping packet at R3");
        comment outimage;

        receivedPackets.update(1);

        END;

END;

repeat;

End;
```

```
comment ********************End of entity classes*******************;


comment ************************main*****************************;

comment ********************distributions************************;

comment ************Getting packet size distribution***************;

packetDist_infile:-new infile("packet_dist.txt");
packetDist_infile.open(Blanks(100));
inf:-packetDist_infile;
readdist(packetLengthDist,"packet_dist");

comment ************************************************************;

interarrivaldist:-new negexp("interArrival",43.6513);
comment This parameter is used to vary traffic load.

generatedPackets:-new Count("Generated packets");
receivedPackets:-new Count("Received packets");
droppedPackets:-new Count("Dropped packets");
packetDelay:-new tally("Delay");

r:-new randint("Random number",1,10);

comment ********************variables*****************************;

bufferSize:=12500;
linkCapacity:= 1000000;

simSeed:=30;

setseed(simSeed);

numReps:=10;
transient_simtime:=2000;
simTime:=10000;

comment ***************Initializing variables**********************;

packetsPerTimeUnit:=0;
packetSum:=0;
averagePacketLength:=0;
averageBitrate:=0;
averageLoad:=0;
averageLoadR2:=0;


WHILE replication <= numReps DO
BEGIN

r_out_q1:-new waitq("r_out_q1");
new ip_router("R2",r_out_q1).schedule(0.0);

r_out_q2:-new waitq("r_out_q2");
new ip_router("R3",r_out_q2).schedule(0.0);

comment ***************Creating packet generator***************;

new packetGenerator("PacketGenerator").schedule(0.0);
```

```
hold(simTime);

replicate;

END;

comment ********************Statistics**************************;

packetsPerTimeUnit:=(packetSum/(simTime-transient_simtime))/numReps;
averagePacketLength:=packetLengthDist.avg;
averageBitrate:=(packetsPerTimeUnit*averagePacketLength*8);
averageLoad:=(averageBitrate/linkCapacity); comment Erlang;

comment ******************Printing statistics*********************;

outtext("Packets per time unit: ");
outimage;
outfix(packetsPerTimeUnit,10,20);
outimage;

outtext("Average packet length: ");
outimage;
outfix(averagePacketLength,10,20);
outimage;

outtext("Average bit rate: ");
outimage;
outfix(averageBitrate,10,20);
outimage;
outtext(" bps");
outimage;

outtext("Link capacity: ");
outimage;
outint(linkCapacity,10);
outimage;
outtext(" bps");
outimage;

outtext("Average load: ");
outimage;
outfix(averageLoad,10,20);
outimage;

end demos;
end;
```

A.1.2 Hyperexponential interarrival distribution

```
BEGIN

external class demos="C:\cim\demos\demos_new.atr";

demos begin

INTEGER bufferSize; comment Bytes;
INTEGER randomNumber;
INTEGER hopCount;
INTEGER i;
INTEGER linkCapacity;
```

```
INTEGER packetSum;
INTEGER simSeed,simTime,transient_simtime,numReps;
REAL packetLength;
REAL averageLoad,packetsPerTimeUnit,averageBitrate;
LONG REAL averagePacketLength;

REF(count) generatedPackets,droppedPackets,receivedPackets;
REF(rdist) packetLengthDist,interArrivalDist1,interArrivalDist2;
REF(tally) packetDelay;
REF(idist) r,random;
REF(waitq) r_out_q1,r_out_q2;
REF(infile) packetDist_infile;


COMMENT **********Classes**********;

COMMENT **********PACKET GENERATOR CLASS**********;

entity CLASS packetGenerator;
Begin

INTEGER sample,i;
i:=1;

WHILE TRUE DO BEGIN

comment **********GENERATING PACKETS DESTINED TO R1**********;

sample:=random.sample;

IF sample=1 THEN BEGIN

        hold(interArrivalDist1.sample);

END ELSE BEGIN

        hold(interArrivalDist2.sample);

END;

        new packet(edit("packet",i),1).schedule(0.0);

IF time > transient_simtime THEN BEGIN

        generatedPackets.update(1);

        packetSum:=packetSum+1;

END;

        hold(interarrivalDist.sample);

        i:=i+1;

END;

End;


COMMENT **********PACKET CLASS**********;
entity CLASS packet(hopCount);
```

```
INTEGER hopCount;

Begin

REAL timestamp;
REAL packetLength,maxSize;

timestamp:=time;


COMMENT ********************PACKET SIZING************************;

maxSize := 1500; COMMENT max ip payload size;

packetLength:= packetLengthDist.sample;

IF packetLength>0 THEN
BEGIN

 IF packetLength > maxSize THEN
 BEGIN

        packetLength := packetLength - maxSize;

 END;

END;


COMMENT ****************end of packet sizing********************;

COMMENT **************PUTTING PACKET IN QUEUE******************;


IF (hopCount=1) THEN BEGIN

IF r_out_q1.length*packetLengthDist.avg < bufferSize THEN BEGIN

        r_out_q1.wait;
        comment outtext("Putting packet in q1");
        comment outimage;

END ELSE BEGIN

        IF time > transient_simtime THEN BEGIN

        droppedPackets.update(1);
        comment outtext("Dropping packet from q1");
        comment outimage;

        END;

END;

END;

IF (hopCount=2) THEN BEGIN

        IF r_out_q2.length*packetLengthDist.avg < bufferSize THEN BEGIN

        r_out_q2.wait;
        comment outtext("Putting packet in q2");
```

```
        comment outimage;

        END ELSE BEGIN

        IF time > transient_simtime THEN BEGIN

                droppedPackets.update(1);
                comment outtext("Dropping packet from q2");
                comment outimage;

        END;

END;

END;

End;

comment ******************IP ROUTER CLASS************************;
entity CLASS ip_router(outQueue);

ref(waitq) outQueue;

Begin

ref(packet) packet_;

LOOP:

packet_:-outQueue.coopt;

IF time>0 THEN BEGIN

        hold(((packet_.packetLength)*8)/(linkCapacity));
        comment Transmission delay;

        comment outtext("Holding for: ");
        comment outfix(((packet_.packetLength)*8)/(linkCapacity),5,10);
        comment outimage;

END;

        packet_.hopCount:=packet_.hopCount+1;


comment *******CHECKING DESTINATION NODE ---
-- 0,7 PROBABILITY OF TRANSIT IN ROUTER 2. IF AT ROUTER 3 - NO TRANSIT********;

IF packet_.hopCount=2 THEN BEGIN

randomNumber:=r.sample;

IF randomNumber>3 THEN BEGIN

packet_.schedule(now);

comment outtext("Transiting packet to R3");
comment outimage;

END ELSE BEGIN

IF time > transient_simtime THEN BEGIN
```

```
        packetDelay.update(time-(packet_.timestamp));

                comment outtext("Dropping packet at R2");
                comment outimage;

                receivedPackets.update(1);

END;

END;

END;

IF packet_.hopCount=3 THEN BEGIN

IF time > transient_simtime THEN BEGIN

        packetDelay.update(time-(packet_.timestamp));

        comment outtext("Dropping packet at R3");
        comment outimage;

receivedPackets.update(1);

END;

END;

repeat;

End;

comment ********************End of entity classes*******************;

comment ************************main*****************************;

comment ********************distributions************************;

comment ************Getting packet size distribution***************;

packetDist_infile:-new infile("packet_dist.txt");
packetDist_infile.open(Blanks(100));
inf:-packetDist_infile;
readdist(packetLengthDist,"packet_dist");

comment ************************************************************;

interArrivalDist1:-new negexp("interArrival",1757.295);
interArrivalDist2:-new negexp("interArrival",35.1461);

generatedPackets:-new Count("Generated packets");
receivedPackets:-new Count("Received packets");
droppedPackets:-new Count("Dropped packets");
packetDelay:-new tally("Delay");

random:-new randint("Random",1,5);
r:-new randint("Random number",1,10);

comment ********************variables****************************;

bufferSize:=12500;
```

```
linkCapacity:= 1000000;

simSeed:=30;

setseed(simSeed);

numReps:=10;
transient_simtime:=2000;
simTime:=10000;

comment **********Initializing variables**********;

packetsPerTimeUnit:=0;
packetSum:=0;
averagePacketLength:=0;
averageBitrate:=0;
averageLoad:=0;
averageLoadR2:=0;


WHILE replication <= numReps DO
BEGIN

r_out_q1:-new waitq("r_out_q1");
new ip_router("R2",r_out_q1).schedule(0.0);

r_out_q2:-new waitq("r_out_q2");
new ip_router("R3",r_out_q2).schedule(0.0);

comment **********Creating packet generator**********;

new packetGenerator("PacketGenerator").schedule(0.0);

hold(simTime);

replicate;

END;


comment **********Statistics**********;

packetsPerTimeUnit:=(packetSum/(simTime-transient_simtime))/numReps;
averagePacketLength:=packetLengthDist.avg;
averageBitrate:=(packetsPerTimeUnit*averagePacketLength*8);
averageLoad:=(averageBitrate/linkCapacity); comment Erlang;

comment **********Printing statistics**********;

outtext("Packets per time unit: ");
outimage;
outfix(packetsPerTimeUnit,10,20);
outimage;

outtext("Average packet length: ");
outimage;
outfix(averagePacketLength,10,20);
outimage;

outtext("Average bit rate: ");
outimage;
outfix(averageBitrate,10,20);
```

```
outimage;
outtext(" bps");
outimage;

outtext("Link capacity: ");
outimage;
outint(linkCapacity,10);
outimage;
outtext(" bps");
outimage;

outtext("Average load: ");
outimage;
outfix(averageLoad,10,20);
outimage;

end demos;
end;
```

A.2 OTN simulator

A.2.1 N.E.D. interarrival distribution

```
BEGIN

external class demos="C:\cim\demos\demos_new.atr";

demos begin

INTEGER bufferSize; comment Bytes;
INTEGER randomNumber;
INTEGER i;
INTEGER simSeed,simTime,transient_simtime,numReps;
INTEGER packetSum;
INTEGER linkCapS1R2,linkCapS1R3;
REAL averagePacketLength,averageLoadS1R2,averageLoadS1R3,
packetsPerTimeUnit,averageBitrate;

REF(count) generatedPackets,droppedPackets,receivedPackets;
REF(rdist) packetLengthDist,interarrivalDist;
REF(tally) packetDelay;
REF(idist) r;
REF(waitq) out_q_to_r2,out_q_to_r3;
REF(infile) packetDist_infile;

COMMENT **********Classes**********;

COMMENT **********PACKET GENERATOR CLASS**********;

entity CLASS packetGenerator;
Begin

INTEGER i;
i:=1;

loop:

randomNumber:=r.sample;

IF (randomNumber > 3) THEN BEGIN

        new packet(edit("packet",i),3).schedule(0.0);

END ELSE BEGIN

        new packet(edit("packet",i),2).schedule(0.0);

END;

IF time > transient_simtime THEN BEGIN

        generatedPackets.update(1);

        packetSum:=packetSum+1;

END;

        hold(interarrivaldist.sample);
```

```
        i:=i+1;

repeat;

End;


COMMENT ********************PACKET CLASS***********************;

entity CLASS packet(destinationNode);

INTEGER destinationNode;

Begin

REAL timestamp;
LONG REAL packetLength,maxSize;

timestamp:=time;


COMMENT ********************PACKET SIZING***********************;

maxSize := 1500; COMMENT max ip payload size;

packetLength:= packetLengthDist.sample;

IF packetLength>0 THEN
BEGIN

        IF packetLength > maxSize THEN
        BEGIN

                packetLength := packetLength - maxSize;

        END;

END;

COMMENT ******************end of packet sizing*********************;

COMMENT ****************PUTTING PACKET IN QUEUE*******************;

IF destinationNode=2 THEN BEGIN

IF out_q_to_r2.length*packetLengthDist.avg < bufferSize THEN BEGIN

        out_q_to_r2.wait;
        comment outtext("Putting packet in out_q_to_r2");
        comment outimage;

END ELSE BEGIN

        IF time > transient_simtime THEN BEGIN

                droppedPackets.update(1);
                comment outtext("Dropping packet from out_q_to_r2");
                comment outimage;

        END;
```

```
END;

END;

IF destinationNode=3 THEN BEGIN

        IF out_q_to_r3.length*packetLengthDist.avg < bufferSize
        THEN BEGIN

                out_q_to_r3.wait;
                comment outtext("Putting packet in out_q_to_r3");
                comment outimage;

        END ELSE BEGIN

        IF time > transient_simtime THEN BEGIN

                droppedPackets.update(1);
                comment outtext("Dropping packet from out_q_to_r3");
                comment outimage;

        END;

END;

END;

End;


comment **************IP ROUTER CLASS*****************;

entity CLASS ip_router(outQ);

ref(waitq) outQ;

Begin

ref(packet) packet_;

LOOP:

packet_:-outQ.coopt;

IF packet_.destinationNode=2 THEN BEGIN

        hold((packet_.packetLength*8)/linkCapS1R2);

        comment outtext("Holding for S1R2: ");
        comment outfix(((packet_.packetLength*8)/linkCapS1R2),5,10);
        comment outimage;

END;


IF packet_.destinationNode=3 THEN BEGIN;

        hold((packet_.packetLength*8)/linkCapS1R3);

        comment outtext("Holding for S1R3: ");
        comment outfix(((packet_.packetLength*8)/linkCapS1R3),5,10);
        comment outimage;
```

```
END;

IF time > transient_simtime THEN BEGIN
        packetDelay.update(time-(packet_.timestamp));
        receivedPackets.update(1);
END;

repeat;
End;

comment ********************End of entity classes*******************;
comment ************************main*****************************;

comment ***********************distributions************************;
comment ************Getting packet size distribution***************;
packetDist_infile:-new infile("packet_dist.txt");
packetDist_infile.open(Blanks(100));
inf:-packetDist_infile;
readdist(packetLengthDist,"packet_dist");

interarrivaldist:-new negexp("interArrival",43.6513);
comment Controls the average bit rate;

generatedPackets:-new Count("Generated packets");
receivedPackets:-new Count("Received packets");
droppedPackets:-new Count("Dropped packets");
packetDelay:-new tally("Delay");

r:-new randint("Random number",1,10);

comment ***********************variables****************************;
linkCapS1R2:=300000;
linkCapS1R3:=700000;

bufferSize:=12500;

simSeed:=30;

setseed(simSeed);

numReps:=10;
transient_simtime:=2000;
simTime:=10000;

comment ***************Initializing variables*********************;
packetsPerTimeUnit:=0;
packetSum:=0;
averagePacketLength:=0;
```

```
averageBitrate:=0;
averageLoadS1R2:=0;
averageLoadS1R3:=0;

WHILE replication <= numReps DO
BEGIN

out_q_to_r2:-new waitq("out_q_to_r2");
new ip_router("R2",out_q_to_r2).schedule(0.0);

out_q_to_r3:-new waitq("out_q_to_r3");
new ip_router("R3",out_q_to_r3).schedule(0.0);

new packetGenerator("PacketGenerator").schedule(0.0);

hold(simTime);

replicate;

END;

packetsPerTimeUnit:=(packetSum/(simTime-transient_simtime))/numReps;
averagePacketLength:=packetLengthDist.avg;
averageBitrate:=(packetsPerTimeUnit*averagePacketLength*8);
averageLoadS1R2:=((averageBitrate*0.3)/linkCapS1R2);
averageLoadS1R3:=((averageBitrate*0.7)/linkCapS1R3);

comment **********Printing statistics**********;

outtext("Packets per time unit to R2/R3: ");
outimage;
outfix(packetsPerTimeUnit,10,20);
outimage;

outtext("Average packet length to R2/R3: ");
outimage;
outfix(averagePacketLength,10,20);
outimage;

outtext("Average bit rate to R2/R3: ");
outimage;
outfix(averageBitrate,10,20);
outimage;
outtext(" bps");
outimage;

outtext("Average load to R2: ");
outimage;
outfix(averageLoadS1R2,10,20);
outimage;

outtext("Average load to R3: ");
outimage;
outfix(averageLoadS1R3,10,20);
outimage;

end demos;
end;
```

A.2.2 Hyperexponential interarrival distribution

```
BEGIN

external class demos="C:\cim\demos\demos_new.atr";

demos begin

INTEGER bufferSize; comment Bytes;
INTEGER randomNumber;
INTEGER i;
INTEGER simSeed,simTime,transient_simtime,numReps;
INTEGER packetSum;
INTEGER linkCapS1R2,linkCapS1R3;
REAL averagePacketLength,averageLoadS1R2,averageLoadS1R3,
packetsPerTimeUnit,averageBitrate;

REF(count) generatedPackets,droppedPackets,receivedPackets;
REF(rdist) packetLengthDist,interArrivalDist1,interArrivalDist2;
REF(tally) packetDelay;
REF(idist) r,random;
REF(waitq) out_q_to_r2,out_q_to_r3;
REF(infile) packetDist_infile;

COMMENT **********Classes**********;

COMMENT **********PACKET GENERATOR CLASS**********;

entity CLASS packetGenerator;
Begin

INTEGER i,sample;
i:=1;

WHILE TRUE DO
BEGIN

sample:=random.sample;

IF sample=1 THEN BEGIN

        hold(interArrivalDist1.sample);

END ELSE BEGIN

        hold(interArrivalDist2.sample);

END;

randomNumber:=r.sample;

IF (randomNumber > 3) THEN BEGIN

        new packet(edit("packet",i),3).schedule(0.0);

END ELSE BEGIN

        new packet(edit("packet",i),2).schedule(0.0);

END;
```

```
IF time > transient_simtime THEN BEGIN

        generatedPackets.update(1);

        packetSum:=packetSum+1;

END;

i:=i+1;

repeat;

End;


COMMENT ********************PACKET CLASS************************;

entity CLASS packet(destinationNode);

INTEGER destinationNode;

Begin

REAL timestamp;
LONG REAL packetLength,maxSize;

timestamp:=time;


COMMENT ********************PACKET SIZING************************;

maxSize := 1500; COMMENT max ip payload size;

packetLength:= packetLengthDist.sample;

IF packetLength>0 THEN
BEGIN

IF packetLength > maxSize THEN
BEGIN

        packetLength := packetLength - maxSize;

END;

 END;

COMMENT ********************end of packet sizing**********************;

COMMENT ********************PUTTING PACKET IN QUEUE*******************;

IF destinationNode=2 THEN BEGIN

        IF out_q_to_r2.length*packetLengthDist.avg < bufferSize THEN BEGIN

        out_q_to_r2.wait;
        comment outtext("Putting packet in out_q_to_r2");
        comment outimage;

        END ELSE BEGIN
```

```
        IF time > transient_simtime THEN BEGIN

                droppedPackets.update(1);
                comment outtext("Dropping packet from out_q_to_r2");
                comment outimage;

        END;

END;

END;

IF destinationNode=3 THEN BEGIN

        IF out_q_to_r3.length*packetLengthDist.avg < bufferSize THEN BEGIN

        out_q_to_r3.wait;
        comment outtext("Putting packet in out_q_to_r3");
        comment outimage;

        END ELSE BEGIN

        IF time > transient_simtime THEN BEGIN

        droppedPackets.update(1);
        comment outtext("Dropping packet from out_q_to_r3");
        comment outimage;

        END;

END;

END;

End;


comment **************IP ROUTER CLASS*****************;

entity CLASS ip_router(outQ);

ref(waitq) outQ;

Begin

ref(packet) packet_;

LOOP:

packet_:-outQ.coopt;

IF packet_.destinationNode=2 THEN BEGIN

        hold((packet_.packetLength*8)/linkCapS1R2);

        comment outtext("Holding for S1R2: ");
        comment outfix(((packet_.packetLength*8)/linkCapS1R2),5,10);
        comment outimage;

END;

IF packet_.destinationNode=3 THEN BEGIN;
```

```
        hold((packet_.packetLength*8)/linkCapS1R3);

        comment outtext("Holding for S1R3: ");
        comment outfix(((packet_.packetLength*8)/linkCapS1R3),5,10);
        comment outimage;

        END;

IF time > transient_simtime THEN BEGIN

        packetDelay.update(time-(packet_.timestamp));

        receivedPackets.update(1);

END;

repeat;

End;


comment ********************End of entity classes*******************;

comment ************************main*****************************;


comment *********************distributions************************;

comment ************Getting packet size distribution***************;

packetDist_infile:-new infile("packet_dist.txt");
packetDist_infile.open(Blanks(100));
inf:-packetDist_infile;
readdist(packetLengthDist,"packet_dist");

interArrivalDist1:-NEW negexp("interArrival",1757.295);
interArrivalDist2:-NEW negexp("interArrival",35.1461);

generatedPackets:-new Count("Generated packets");
receivedPackets:-new Count("Received packets");
droppedPackets:-new Count("Dropped packets");
packetDelay:-new tally("Delay");

random:-NEW randint("Random",1,5);
r:-new randint("Random number",1,10);

comment *********************variables****************************;

linkCapS1R2:=300000;
linkCapS1R3:=700000;

bufferSize:=12500;

simSeed:=30;

setseed(simSeed);

numReps:=10;
transient_simtime:=2000;
simTime:=10000;
```

```
comment ***************Initializing variables*********************;

packetsPerTimeUnit:=0;
packetSum:=0;
averagePacketLength:=0;
packetLengthCounter:=0;
averageBitrate:=0;
averageLoadS1R2:=0;
averageLoadS1R3:=0;

WHILE replication <= numReps DO
BEGIN

out_q_to_r2:-new waitq("out_q_to_r2");
new ip_router("R2",out_q_to_r2).schedule(0.0);

out_q_to_r3:-new waitq("out_q_to_r3");
new ip_router("R3",out_q_to_r3).schedule(0.0);

new packetGenerator("PacketGenerator").schedule(0.0);

hold(simTime);

replicate;

END;

packetsPerTimeUnit:=(packetSum/(simTime-transient_simtime))/numReps;
averagePacketLength:=packetLengthDist.avg;
averageBitrate:=(packetsPerTimeUnit*averagePacketLength*8);
averageLoadS1R2:=((averageBitrate*0.3)/linkCapS1R2);
averageLoadS1R3:=((averageBitrate*0.7)/linkCapS1R3);

comment *****************Printing statistics*********************;

outtext("Packets per time unit to R2/R3: ");
outimage;
outfix(packetsPerTimeUnit,10,20);
outimage;

outtext("Average packet length to R2/R3: ");
outimage;
outfix(averagePacketLength,10,20);
outimage;

outtext("Average bit rate to R2/R3: ");
outimage;
outfix(averageBitrate,10,20);
outimage;
outtext(" bps");
outimage;

outtext("Average load to R2: ");
outimage;
outfix(averageLoadS1R2,10,20);
outimage;

outtext("Average load to R3: ");
outimage;
outfix(averageLoadS1R3,10,20);
outimage;
```

```
end demos;
end;
```

A.3 Reconfigurable OTN simulator

A.3.1 N.E.D. interarrival distribution

```
BEGIN

external class demos="C:\cim\demos\demos_new.atr";

demos begin

INTEGER bufferSize; comment Bytes;
INTEGER randomNumber;
INTEGER i;
INTEGER simSeed,simTime,transient_simtime,numReps;
INTEGER packetSum;
INTEGER linkCapS1R2,linkCapS1R3;
REAL averagePacketLength,averageLoadS1R2,averageLoadS1R3,
packetsPerTimeUnit,averageBitrate;

REF(count) generatedPackets,droppedPackets,receivedPackets;
REF(rdist) packetLengthDist,interarrivalDist;
REF(tally) packetDelay;
REF(idist) r;
REF(waitq) out_q_to_r2,out_q_to_r3;
REF(infile) packetDist_infile;

COMMENT **********Classes**********;

COMMENT **********PACKET GENERATOR CLASS**********;

entity CLASS packetGenerator;
Begin

INTEGER i;
i:=1;

loop:

randomNumber:=r.sample;

IF (randomNumber > 3) THEN BEGIN

        new packet(edit("packet",i),3).schedule(0.0);

END ELSE BEGIN

        new packet(edit("packet",i),2).schedule(0.0);

END;

IF time > transient_simtime THEN BEGIN

        generatedPackets.update(1);

        packetSum:=packetSum+1;
```

```
END;

        hold(interarrivaldist.sample);

        i:=i+1;

repeat;

End;


COMMENT ********************PACKET CLASS************************;

entity CLASS packet(destinationNode);

INTEGER destinationNode;

Begin

REAL timestamp;
LONG REAL packetLength,maxSize;

timestamp:=time;


COMMENT ********************PACKET SIZING************************;

maxSize := 1500; COMMENT max ip payload size;

packetLength:= packetLengthDist.sample;

IF packetLength>0 THEN
BEGIN

        IF packetLength > maxSize THEN
        BEGIN

                packetLength := packetLength - maxSize;

        END;

        END;


COMMENT ******************end of packet sizing*********************;

COMMENT ****************PUTTING PACKET IN QUEUE*******************;

IF destinationNode=2 THEN BEGIN

IF out_q_to_r2.length*packetLengthDist.avg < bufferSize THEN BEGIN

        out_q_to_r2.wait;
        comment outtext("Putting packet in out_q_to_r2");
        comment outimage;

        END ELSE BEGIN

        IF time > transient_simtime THEN BEGIN

        droppedPackets.update(1);
```

```
        comment outtext("Dropping packet from out_q_to_r2");
        comment outimage;

END;

END;

END;

IF destinationNode=3 THEN BEGIN

IF out_q_to_r3.length*packetLengthDist.avg < bufferSize THEN BEGIN

        out_q_to_r3.wait;
        comment outtext("Putting packet in out_q_to_r3");
        comment outimage;

END ELSE BEGIN

        IF time > transient_simtime THEN BEGIN

        droppedPackets.update(1);
        comment outtext("Dropping packet from out_q_to_r3");
        comment outimage;

END;

END;

END;

End;

comment **************IP ROUTER CLASS*****************;

entity CLASS ip_router(outQ);

ref(waitq) outQ;

Begin

ref(packet) packet_;

LOOP:

packet_:-outQ.coopt;

IF packet_.destinationNode=2 THEN BEGIN

IF time > transient_simtime THEN BEGIN

IF out_q_to_r2.length > out_q_to_r3.length THEN BEGIN

        IF linkCapS1R3 > 500000 THEN BEGIN

        linkCapS1R2:=linkCapS1R2+1000;
        linkCapS1R3:=linkCapS1R3-1000;

        END;

END;
```

```
END;

hold((packet_.packetLength*8)/linkCapS1R2);

comment outtext("Holding for S1R2: ");
comment outfix(((packet_.packetLength*8)/linkCapS1R2),5,10);
comment outimage;

END;

IF packet_.destinationNode=3 THEN BEGIN

IF time > transient_simtime THEN BEGIN

        IF out_q_to_r3.length > out_q_to_r2.length THEN BEGIN

        IF linkCapS1R2 > 250000 THEN BEGIN

                linkCapS1R3:=linkCapS1R3+1000;
                linkCapS1R2:=linkCapS1R2-1000;

END;

END;

END;

hold((packet_.packetLength*8)/linkCapS1R3);

comment outtext("Holding for S1R3: ");
comment outfix(((packet_.packetLength*8)/linkCapS1R3),5,10);
comment outimage;

END;


IF time > transient_simtime THEN BEGIN

        packetDelay.update(time-(packet_.timestamp));

        receivedPackets.update(1);

END;


repeat;

End;


comment ********************End of entity classes*******************;

comment *************************main*****************************;


comment ********************distributions************************;

comment ************Getting packet size distribution***************;

packetDist_infile:-new infile("packet_dist.txt");
packetDist_infile.open(Blanks(100));
inf:-packetDist_infile;
```

```
readdist(packetLengthDist,"packet_dist");

interarrivaldist:-new negexp("interArrival",43.6513);
comment Controls the bit rate;

generatedPackets:-new Count("Generated packets");
receivedPackets:-new Count("Received packets");
droppedPackets:-new Count("Dropped packets");
packetDelay:-new tally("Delay");

r:-new randint("Random number",1,10);


comment **********variables**********;

linkCapS1R2:=300000;
linkCapS1R3:=700000;

bufferSize:=12500;

simSeed:=30;

setseed(simSeed);

numReps:=10;
transient_simtime:=2000;
simTime:=10000;

comment **********Initializing variables**********;

packetsPerTimeUnit:=0;
packetSum:=0;
averagePacketLength:=0;
packetLengthCounter:=0;
averageBitrate:=0;
averageLoadS1R2:=0;
averageLoadS1R3:=0;


WHILE replication <= numReps DO
BEGIN

out_q_to_r2:-new waitq("out_q_to_r2");
new ip_router("R2",out_q_to_r2).schedule(0.0);

out_q_to_r3:-new waitq("out_q_to_r3");
new ip_router("R3",out_q_to_r3).schedule(0.0);

new packetGenerator("PacketGenerator").schedule(0.0);

hold(simTime);

outtext("S1R2: ");
outimage;
outfix(linkCapS1R2,10,20);
outimage;

outtext("S1R3: ");
outimage;
outfix(linkCapS1R3,10,20);
outimage;
```

```
replicate;

END;

packetsPerTimeUnit:=(packetSum/(simTime-transient_simtime))/numReps;
averagePacketLength:=packetLengthDist.avg;
averageBitrate:=(packetsPerTimeUnit*averagePacketLength*8);
averageLoadS1R2:=((averageBitrate*0.3)/linkCapS1R2);
averageLoadS1R3:=((averageBitrate*0.7)/linkCapS1R3);

comment **********Printing statistics**********;

outtext("Packets per time unit to R2/R3: ");
outimage;
outfix(packetsPerTimeUnit,10,20);
outimage;

outtext("Average packet length to R2/R3: ");
outimage;
outfix(averagePacketLength,10,20);
outimage;

outtext("Average bit rate to R2/R3: ");
outimage;
outfix(averageBitrate,10,20);
outimage;
outtext(" bps");
outimage;

outtext("Average load to R2: ");
outimage;
outfix(averageLoadS1R2,10,20);
outimage;

outtext("Average load to R3: ");
outimage;
outfix(averageLoadS1R3,10,20);
outimage;


end demos;
end;
```

A.3.2 Hyperexponential interarrival distribution

```
BEGIN

external class demos="C:\cim\demos\demos_new.atr";

demos begin

INTEGER bufferSize; comment Bytes;
INTEGER randomNumber;
INTEGER i;
INTEGER simSeed,simTime,transient_simtime,numReps;
INTEGER packetSum;
INTEGER linkCapS1R2,linkCapS1R3;
REAL averagePacketLength,averageLoadS1R2,averageLoadS1R3,
packetsPerTimeUnit,averageBitrate;
```

```
REF(count) generatedPackets,droppedPackets,receivedPackets;
REF(rdist) packetLengthDist,interArrivalDist1,interArrivalDist2;
REF(tally) packetDelay;
REF(idist) r,random;
REF(waitq) out_q_to_r2,out_q_to_r3;
REF(infile) packetDist_infile;

COMMENT **********Classes**********;

COMMENT **********PACKET GENERATOR CLASS**********;

entity CLASS packetGenerator;
Begin

INTEGER i,sample;
i:=1;

WHILE TRUE DO BEGIN

sample:=random.sample;

IF sample=1 THEN BEGIN

        hold(interArrivalDist1.sample);

END ELSE BEGIN

        hold(interArrivalDist2.sample);

END;

randomNumber:=r.sample;

IF (randomNumber > 3) THEN BEGIN

        new packet(edit("packet",i),3).schedule(0.0);

END ELSE BEGIN

        new packet(edit("packet",i),2).schedule(0.0);

END;

IF time > transient_simtime THEN BEGIN

        generatedPackets.update(1);

        packetSum:=packetSum+1;

END;

i:=i+1;

repeat;

End;

COMMENT **********PACKET CLASS**********;

entity CLASS packet(destinationNode);
```

```
INTEGER destinationNode;

Begin

REAL timestamp;
LONG REAL packetLength,maxSize;

timestamp:=time;


COMMENT ********************PACKET SIZING***********************;

maxSize := 1500; COMMENT max ip payload size;

packetLength:= packetLengthDist.sample;

IF packetLength>0 THEN
BEGIN

IF packetLength > maxSize THEN
BEGIN

        packetLength := packetLength - maxSize;

END;

END;


COMMENT ********************end of packet sizing*********************;

COMMENT ****************PUTTING PACKET IN QUEUE*******************;

IF destinationNode=2 THEN BEGIN

IF out_q_to_r2.length*packetLengthDist.avg < bufferSize THEN BEGIN

        out_q_to_r2.wait;
        comment outtext("Putting packet in out_q_to_r2");
        comment outimage;

END ELSE BEGIN

IF time > transient_simtime THEN BEGIN

        droppedPackets.update(1);
        comment outtext("Dropping packet from out_q_to_r2");
        comment outimage;

END;

END;

END;

IF destinationNode=3 THEN BEGIN

IF out_q_to_r3.length*packetLengthDist.avg < bufferSize THEN BEGIN

        out_q_to_r3.wait;
        comment outtext("Putting packet in out_q_to_r3");
        comment outimage;
```

```
END ELSE BEGIN

IF time > transient_simtime THEN BEGIN

        droppedPackets.update(1);
        comment outtext("Dropping packet from out_q_to_r3");
        comment outimage;

END;

END;

END;

End;

comment **************IP ROUTER CLASS*****************;

entity CLASS ip_router(outQ);

ref(waitq) outQ;

Begin

ref(packet) packet_;

LOOP:

packet_:-outQ.coopt;

IF packet_.destinationNode=2 THEN BEGIN

IF time > transient_simtime THEN BEGIN

IF out_q_to_r2.length > out_q_to_r3.length THEN BEGIN

        IF linkCapS1R3 > 500000 THEN BEGIN

        linkCapS1R2:=linkCapS1R2+1000;
        linkCapS1R3:=linkCapS1R3-1000;

        END;

END;

END;

hold((packet_.packetLength*8)/linkCapS1R2);

comment outtext("Holding for S1R2: ");
comment outfix(((packet_.packetLength*8)/linkCapS1R2),5,10);
comment outimage;

END;

IF packet_.destinationNode=3 THEN BEGIN

IF time > transient_simtime THEN BEGIN

        IF out_q_to_r3.length > out_q_to_r2.length THEN BEGIN
```

```
        IF linkCapS1R2 > 250000 THEN BEGIN

                linkCapS1R3:=linkCapS1R3+1000;
                linkCapS1R2:=linkCapS1R2-1000;

        END;

END;

END;

hold((packet_.packetLength*8)/linkCapS1R3);

comment outtext("Holding for S1R3: ");
comment outfix(((packet_.packetLength*8)/linkCapS1R3),5,10);
comment outimage;

END;


IF time > transient_simtime THEN BEGIN

        packetDelay.update(time-(packet_.timestamp));

        receivedPackets.update(1);

END;


repeat;

End;

comment ********************End of entity classes*******************;

comment *************************main*****************************;


comment *********************distributions************************;

comment ************Getting packet size distribution***************;

packetDist_infile:-new infile("packet_dist.txt");
packetDist_infile.open(Blanks(100));
inf:-packetDist_infile;
readdist(packetLengthDist,"packet_dist");

interArrivalDist1:-new negexp("interArrival",1757.295);
interArrivalDist2:-new negexp("interArrival",35.1461);

generatedPackets:-new Count("Generated packets");
receivedPackets:-new Count("Received packets");
droppedPackets:-new Count("Dropped packets");
packetDelay:-new tally("Delay");

random:-new randint("Random",1,5);
r:-new randint("Random number",1,10);


comment ********************variables****************************;

linkCapS1R2:=300000;
```

```
linkCapS1R3:=700000;

bufferSize:=12500;

simSeed:=30;

setseed(simSeed);

numReps:=10;
transient_simtime:=2000;
simTime:=10000;

comment ***************Initializing variables*********************;

packetsPerTimeUnit:=0;
packetSum:=0;
averagePacketLength:=0;
packetLengthCounter:=0;
averageBitrate:=0;
averageLoadS1R2:=0;
averageLoadS1R3:=0;


WHILE replication <= numReps DO
BEGIN

out_q_to_r2:-new waitq("out_q_to_r2");
new ip_router("R2",out_q_to_r2).schedule(0.0);

out_q_to_r3:-new waitq("out_q_to_r3");
new ip_router("R3",out_q_to_r3).schedule(0.0);

new packetGenerator("PacketGenerator").schedule(0.0);

hold(simTime);

outtext("S1R2: ");
outimage;
outfix(linkCapS1R2,10,20);
outimage;

outtext("S1R3: ");
outimage;
outfix(linkCapS1R3,10,20);
outimage;

replicate;

END;

packetsPerTimeUnit:=(packetSum/(simTime-transient_simtime))/numReps;
averagePacketLength:=packetLengthDist.avg;
averageBitrate:=(packetsPerTimeUnit*averagePacketLength*8);
averageLoadS1R2:=((averageBitrate*0.3)/linkCapS1R2);
averageLoadS1R3:=((averageBitrate*0.7)/linkCapS1R3);

comment ***************Printing statistics********************;

outtext("Packets per time unit to R2/R3: ");
outimage;
outfix(packetsPerTimeUnit,10,20);
outimage;
```

```
outtext("Average packet length to R2/R3: ");
outimage;
outfix(averagePacketLength,10,20);
outimage;

outtext("Average bit rate to R2/R3: ");
outimage;
outfix(averageBitrate,10,20);
outimage;
outtext(" bps");
outimage;

outtext("Average load to R2: ");
outimage;
outfix(averageLoadS1R2,10,20);
outimage;

outtext("Average load to R3: ");
outimage;
outfix(averageLoadS1R3,10,20);
outimage;


end demos;
end;
```

Appendix B

DEMOS library changes

B.1 demos.atr and demos_new.atr difference

A procedure named avg is added to the DEMOS library in order to calculate the average packet length.

Code added to line 826 in demos_new.atr:

```
PROCEDURE Avg IS long real procedure Avg;;
```

Code added to lines 986-1000 in demos_new.atr:

```
long real procedure Avg;

begin long real Q;
integer K;

        K:=1;
        while K < Size do begin

                K:=K+1;
                Q:=Q + (X(K)+X(K-1))/2*(P(K)-P(K-1));

    end;

    Avg:=Q;

end***Avg***;
```

Appendix C

Input parameter file

C.1 Packet size distribution

```
packet_dist EMPIRICAL 5

40    0.00
44    0.62
552   0.75
576   0.83
1500  1.00
```