SCALABLE HIGH-CAPACITY HIGH-FAN-OUT OPTICAL NETWORKS FOR CONSTRAINED ENVIRONMENTS

by

Syed Ali Haider

A dissertation submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in
Optical Science and Engineering

Charlotte

2012

ABSTRACT

SYED ALI HAIDER. Scalable high-capacity high-fan-out optical networks for constrained environments. (Under the direction of Dr. M YASIN AKHTAR RAJA)

The investigations carried out as part of the dissertation address the architecture and application of optical access networks pertaining to high-capacity and high fan-out applications such as in-flight entertainment (IFE) and video-gaming environment. High-capacity and high-fan-out optical networks have a multitude of applications such as expo-centers, train area networks (TAN), video gaming competitions and other applications that require large number of connected users. For the purpose of keeping the scope of the dissertation within limit however, we have concentrated this work on IFE systems. IFE systems present unique challenges at physical and application layers alike. In-flight entertainment (IFE) systems have been a part of passengers' experience for a while now. Currently available systems can be considered a bare-bone at best due to lack of adequate performance and support infrastructure. According to electronic arts (EA) – one of the largest developers of video games in the world, an increase in demand for electronically distributed video games will exceed boxed games in just a matter of few years. This also shows a shifting trend towards the electronic distribution of video game content as opposed to physical distribution.

Against the same backdrop, the dissertation project involved defining a novel system architecture and capacity based on the requirements for development of novel physical layer architecture utilizing optical networks for high-speed and high-fan-out distribution of content. At the physical layer of the stacked communication model a novel high-fan-out optical network was proposed and simulated for high data-rates. Having defined the physical layer, protocol stack was identified through rigorous observations and data traffic analysis from a large set of traffic traces obtained from various sources in order to understand the distribution and behavior of video game related traffic compared with regular internet traffic. Data requirements were laid down based on analysis keeping in mind that bandwidth requirements are increasing at a tremendous pace and that the network should be able to support future high-definition and 3D gaming as well. Based on the data analysis, analytical models and latency analysis models were also developed for bandwidth allocation in the high-fan-out network architectures. Analytical modeling gives an insight into the performance of the technique as a function of incoming traffic whereas latency analysis exposes the delay factors involved in running the technique over time. “State-full bandwidth allocation” (SBA) was proposed as part of the network layer design for upstream transmission. The novel technique involves keeping state information from previous states for future allocation.

The results show that the proposed high-fan-out high-capacity physical layer architecture can be used to distribute video-gaming related content. Also, latency analysis and design and development of a novel SBA algorithm were carried out. Results were quiet promising, in that; a large number of users can be supported on the same single channel network. SBA criteria can be applied to multi-channel networks such as the physical architecture proposed / simulated and investigated in this project. In summary, the project involved design of a novel physical layer; network layer and protocol stack of the communication model and verification by simulations and mathematical modeling while adhering to application layer requirements.

ACKNOWLEDGMENT

First of all, I would like to thank Almighty God for blessing me with the opportunity and persistence to complete this work. I am humbled by what I have learned throughout the course of this degree and from the environment around me. I would like to thank Dr Khurram Kazi (NYIT) whose valuable comments and suggestion led the project in the right direction and his critique was the key factor in finishing this work. I would like to thank Dr. M. Yasin Akhtar Raja, my advisor, for steering the project and for his guidance and time during the course of this work and other related projects. Exposure that I got by working with him in the optical networks lab and with annual conference HONET is simply invaluable. I would also extend my sincere gratitude to my dissertation committee members, Dr. Michael A. Fiddy, Dr Linda Xie, Dr. Greg Gbur and Dr Jamie Payton for their valuable comments on the dissertation. Special thanks to Dr. Arshad Ali and Dr. S. M. H. Zaidi at National University of Sciences and Technology (NUST), School of Electrical Engineering and Computer Science (SEECS) for recommending me for this opportunity. I am also indebted to the department of Physics and Optical Science for continued support and help in professional growth in the area of teaching and learning beyond my research area. I would also recognize my fellow graduate students and friends who made my graduate study at UNC Charlotte very memorable period of life. Last but not the least; I would like to acknowledge the professional and Graduate Assistant Scholarship Program (GASP) support from the Graduate School at the UNC Charlotte and support extended by International Student Scholar Office (ISSO).

DEDICATION

To my parents Maj. (r) Syed Sajjad Haider and Maj. (r) Dr. Zarafat Haider.

TABLE OF CONTENTS

LIST OF TABLES ix

LIST OF FIGURES x

LIST OF ABBREVIATIONS xii

CHAPTER 1: INTRODUCTION TO IN-FLIGHT ENTERTAINMENT (IFE) SYSTEMS 1

1.1 The Concept/Paradigm 5

1.2 Requirements 6

1.3 Service oriented architectures 10

CHAPTER 2: PHYSICAL LAYER 12

2.1 Physical layer technologies 14

2.1.1 Overview of OFDM 15

2.1.2 Overview of TDM / Single channel 18

2.2 Proposed architecture 19

2.2.1 Details of Network Segment used in Simulations 19

2.3 Experiments 23

2.3.1 Simulation software 24

2.3.2 Results 25

2.4 Discussions and relevant remarks 28

CHAPTER 3: PROTOCOL STACK 29

3.1 Initial considerations 29

3.2 Real-time Video Game Stream (RVGS) 34

3.3 Data and observations 37

3.4 Discussions and remarks 44

CHAPTER 4: BANDWIDTH ALLOCATION 46

4.1 Data collection and the datasets used 49

4.2 Bandwidth allocation for PON 50

4.3 State-full Bandwidth Allocation (SBA) for High-fan-out Optical Networks 54

4.4 Experiments 56

4.4.1 Analytical Model 56

4.4.2 Latency Analysis 68

4.4.3 Results 74

4.5 Discussions 81

CHAPTER 5: DISCUSSIONS AND CONCLUDING REMARKS 83

5.1 Concluding remarks 83

5.2 Future work 85

REFERENCES 88

APPENDIX A: RC-DBA CODE 93

APPENDIX B: RC-DBA & EFDBA WAITING TIMES CODE 97

APPENDIX C: SBA CODE 99

APPENDIX D: SBA WAITING TIME CODE 104

APPENDIX E: DATA CAPTURE CODE 106

LIST OF TABLES

TABLE 1: Packet size UDP/IP stack 32

TABLE 2: Datasets/Traces 42

TABLE 3: Weight table – SBA 55

TABLE 4: Parameters 58

TABLE 5: RC-DBA 71

LIST OF FIGURES

FIGURE 1: System Deployment – Optical network based architecture for in-flight systems (adapted from flight schematic) 6

FIGURE 2: OFDM sub-carriers (frequency domain) to time-domain signal 15

FIGURE 3: OFDM signal 17

FIGURE 4: A square wave generated with multiple sine waves 18

FIGURE 5: Typical time division multiplexing (TDM) allocation cycle 19

FIGURE 6: Block Diagram of proposed architecture 20

FIGURE 7: Component level view of proposed architecture 24

FIGURE 8: Coherent 100Gb/s transmitter with 2bits/symbol 25

FIGURE 9: 100Gb/s receiver 26

FIGURE 10: Effect of increasing the number of subcarriers on BER 27

FIGURE 11: Received signal power and its effect on BER performance for the 16-QAM OFDM signal 27

FIGURE 12: Headers information added by each protocol layer 30

FIGURE 13: UDP/IP protocol stack on Ethernet channel 33

FIGURE 14: Real-time Video Game Stream (RVGS) packet header and data 35

FIGURE 15: Data extracts from CAIDA datasets 40

FIGURE 16: Session information extracted from datasets 41

FIGURE 17: Video game datasets for SSR and CSR based games 44

FIGURE 18: SSR vs CSR 48

FIGURE 19: Video game reference traces 50

FIGURE 20a: Requests from connected ONUs 52

FIGURE 20b: An overview of RC-DBA ratio counter and allocation table 52

FIGURE 21: Cycle time distribution in RC-DBA 53

FIGURE 22: Bandwidth allocation in SBA 59

FIGURE 23: A dynamic cycle set 60

FIGURE 24a: Flow chart for allocation in SBA 61

FIGURE 24b: Flow chart: first and second allocation 62

FIGURE 25: Changing cycle set 64

FIGURE 26a: Residual time after every cycle for 100 cycles for random users 66

FIGURE 26b: Residual time after every cycle for 500 cycles for random users 67

FIGURE 26c: Residual time after every cycle for 500 cycles for 250 users 67

FIGURE 27: Residual time calculation 72

FIGURE 28: Waiting time increases with larger cycle time (T) 75

FIGURE 29: Waiting time increases with increase in arrival rate at ONUs 75

FIGURE 30a: Waiting time vs Arrival rate for Poisson arrivals 76

FIGURE 30b: Waiting time vs Arrival rate for Poisson based arrivals (TDMA) 76

FIGURE 31a: Waiting time vs Offered load for (N=64) Pareto arrivals (RC-DBA) 77

FIGURE 31b: Waiting time vs Offered load for (N=32) Pareto arrivals (RC-DBA) 78

FIGURE 31c: Waiting time vs Offered load for (N=16) Pareto arrivals (RC-DBA) 78

FIGURE 32: Comparison of waiting time and residual time 79

FIGURE 33: Waiting time increases with increase in number of active users 80

FIGURE 34: First allocation time dependence on number of users and associated waiting time 81

FIGURE 35: Waiting time increases with increase in cycle time 81

FIGURE 36: Applications of high-capacity high-fan-out optical networks 87

LIST OF ABBREVIATIONS

ASK	Amplitude Shift Keying
BA	Bandwidth Allocation
BER	Bit Error Rate
CAIDA	Corporate Association for Internet Data Analysis
CO-OFDM	Coherent Optical OFDM
CSR	Client Side Rendering
DBA	Dynamic Bandwidth Allocation
EA	Electronic Arts
EDFA	Erbium-doped Fiber Amplifier
EFDBA	Enhanced Fair DBA
EW	Ensured Window
FEC	Forward Error Correction
FFT	Fast Fourier Transform
HD	High Definition (720p)
ICT	Information and Communication Technology
IFE	In-flight Entertainment
IFES	In-flight Entertainment System
IFFT	Inverse FFT
IP	Internet Protocol
ISI	Inter-symbol Interference
LAN	Local Area Network
LO	Local Oscillator

LPF	Low Pass Filter
MTU	Maximum Transmission Unit
MZM	Mach-Zehnder Modulator
NIC	Network Interface Card
OC	Optical Carrier
OFDM	Orthogonal Frequency Division Multiplexing
OLT	Optical Line Terminal
ON	Optical Network
ONU/ONT	Optical Network Unit/ Optical Network Terminal
OOK	On-Off Keying
PC	Power Combiner
PIN	P-type Intrinsic N-type
PM-QPSK	Polarization Multiplexing QPSK
PON	Passive Optical Network
PRBG	Pseudo Random Bit Generator
QAM	Quadrature Amplitude Modulation
QPSK	Quadrature Phase Shift keying
RC-DBA	Ratio Counter based DBA
ROTW	Rest Of The Window
RVGS	Real-time Video Game Stream
SBA	State-full Bandwidth Allocation
SLA	Service Level Agreement
SSR	Server Side Rendering

TCP	Transmission Control Protocol
TDM	Time Division Multiplexing
UDP	User Datagram Protocol
VoIP	Voice over Internet Protocol
WAEA	World Airline Entertainment Association

CHAPTER 1: INTRODUCTION TO IN-FLIGHT ENTERTAINMENT (IFE) SYSTEMS

Entertainment industry in general and the video game industry in particular is growing at a faster pace than ever before owing to information and communication technologies (ICT). A tremendous digital storage capacity, high-speed transport and optical networks enhanced mobility is an impetuous in the growth of such sector. Conventional video gaming has been steadily growing every year and since introduction of modern gaming consoles such as Xbox and PlayStation, industry has seen unprecedented growth. Not only in conventional gaming, the video games have started adopting cloud based paradigm but also there are services that offer cloud based gaming [1]. This trend is endorsed by the Electronic Arts (EA) a leading video game developer [2]. EA predicts that in a matter of years, sales of electronically downloaded games will outnumber boxed games. This prediction is based on the statistical data and facts and it opens up potentially many new arenas that can benefit from the advancement in the video game industry. Especially, high-speed communication systems can be developed that offer services that are unprecedented in this industry.

While there are a large variety of video games spanning all walks and aspects of our lives, we will concentrate on the in-flight games and their requirements. We focus on in-flight entertainment (IFE) systems as one illustrated in Figure 1. Other applications that can be served with the proposed solutions are explained in chapter 5. An IFE system is a specialized communication network on-board a commercial airliner that provides entertainment services to passengers. Entertainment can be in the form of on-demand video, internet based services and video gaming. We focus on video gaming as the required service. A simple implementation of such a network using a multi-level optical fiber based network is shown in Figure 1. There is a tremendous opportunity in developing the in-flight gaming/entertainment infrastructure and resources as new display and interaction technologies make significant strides. Also, as the processing power increases, more powerful gaming engines are developed. The end result of course is to provide users/customers with an enhanced video gaming experience on-board commercial airliners' fleet. Long-haul flights are the best environment that can utilize the state-of-the-art video gaming systems because passengers have more time for longer game play. This need becomes even more pronounced when we consider the number of hours spent in trans-continental and oceanic flights carrying everyday millions of passengers [3] around the world. However, in order to have a high-fidelity, high-performance, and high reliability infrastructure on-board, there needs to be a state-of-the-art underlying network architecture to support the systems in the aircrafts.

Optical networks owing to their ultra-high bandwidth have the capability to bring high-quality gaming experience to the users/passengers by virtue of high-speed data transfer. Optical networks have other benefits too for instance; they are light weight; offer extremely high capacity; are highly secure and non-intrusive due to the nature of the optical fiber; generally do not require too much maintenance and have a future proof infrastructure. Energy efficiency is inherent to the optical networks and being a wired infrastructure thereby does not interfere with existing flight critical communications. Along with the above mentioned benefits, due to high-speed, the latency is better than competitor technologies such as copper based networks or Wi-Fi. Copper based networks are heavy with heavier equipment and wireless networks may interfere with flight critical communication and offer interrupted service.

The scope of this dissertation project is three pronged;

i) In the first phase, at the physical layer level, novel optical network architecture for high-fan-out systems was designed and simulated on industry standard simulation software Optsim® by RSoft. “High-fan-out” refers to the ability to support a large number of simultaneous users. Proposed architecture uses orthogonal frequency division multiplexing (OFDM) and utilizes coherent detection. Having multiple sub-carriers allows the designers/developers to provide bandwidth to a large number of users much more efficiently and without any bandwidth wastage. We also use recently demonstrated high-capacity networks [4, 5] with direct detection, as the underlying network for our TDM based bandwidth allocation introduced and discussed in part (iii).

ii) Second phase of the project involved identification of a protocol stack and definition of real-time video game stream (RVGS) for transporting the video game related data over the network. Since data is transmitted in a unique way depending on what service is being used or the system architecture we define the primitives and communication stack for two types of video gaming paradigms involving distinct experiences. The two types of experiences are categorized as:

   a. Server side rendering

   b. Client side rendering

Briefly, the server side rendering (SSR) systems have a huge processing power at the server end (or service provider end) and frame rendering occurs at the servers. Finished frames are then transmitted to the users much like a video transmission system except that the experience is largely interactive. OnLive [1] is one example of a server side rendering based video gaming system. The client side rendering (CSR) system on the other hand is a conventional video game playing system where clients install the game engines and requisite mapping information. Only vectors are transmitted over the network. A vector contains information about motion of a certain object inside a video game, for instance, information about players' new location or change in players' viewing angle may be transmitted as a vector in the form of change graphic coordinates. Since the characters in video games are also made up of a mesh of known structures change vectors only change the locations on characters (mesh) on the map. Rendered image is shown to the user as a continuous video called the game play. This process can be pre-coded as well [6]. One vector represents one such change and for the whole game play millions of vectors are transmitted. Video games are very complex software systems and require immense choreography when the game is being played. Vectors present a lightweight transportation envelops that can be sent across multiple players so that all screens remain synchronized. PlayStation and Xbox are examples of CSR system.

iii) Third phase of the project involved design and development of an efficient bandwidth allocation algorithm that essentially serves as a soul for the distribution network while catering to a large user base and keeping the latency within a minimum acceptable range. We developed novel bandwidth allocation algorithms for direct detection single carrier systems that also works with OFDM based architecture [7].

It can be seen that the three pronged project involves design and development of the system starting for the physical layer and reaching up to the application layer, for the source constrained video gaming environments such as IFE. Data collection for design and model validation was a major phase of the project. Most datasets used for observation and analysis are real-life datasets obtained from CAIDA [8] and OnLive [1].

1.1 The Concept/Paradigm

Airports in the United States handle over 4000 (~ 4277 to be exact) flights every day [3]. Millions of people travel through the system each day. There exists a huge opportunity for the airline industry to tap in to such opportunity of dedicated audience/clients. State-of-the-art video gaming systems can help enhance traveling experience for a large number of commuters and leisure travelers alike who enjoy video games and associated interactive technologies. The World Airline Entertainment Association (WAEA) has called on-board video gaming system a critical part of future of in-flight entertainment in 2002 [9]. Not a major breakthrough, but there has been some progress lately on this front. The key issue with the existing systems [10-12] is the lack of resilience, physical infrastructure and operational means that can support the demanding video gaming environment of today. The systems are also not standardized. On current systems, video games are basic in nature and system uptime is low. There is a need for support for high-quality graphical contents for user/passengers. A basic and simple theme of such an optical fiber based content distribution system is shown in Figure 1 that presents salient features; its architectural details are discussed in later sections and chapters.

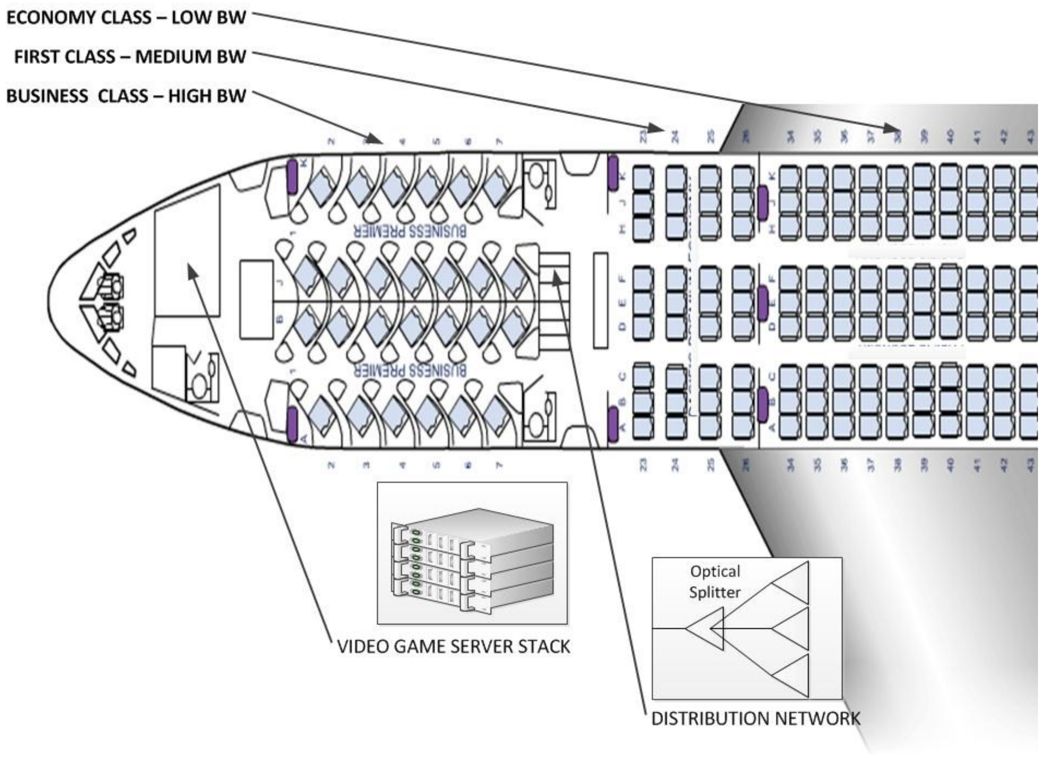


Figure 1: System Deployment – Optical network based architecture for in-flight systems (adapted from flight schematic) [13]

Next, the basic requirements for developing a high-speed video gaming network for on-board use in an aircraft are discussed briefly in the sub-section below.

1.2 Requirements

When looking carefully, in-flight systems in general are constrained by requirements that are unparalleled in any other industry. Airworthiness standards dictate a strict set of rules for equipment to be placed on-board a commercial airliner [14]. Besides the security and challenges related to interference with flight-critical communication systems, high-resolution graphics and low-latency are considered to be the two most important characteristics of a good gaming experience and therefore following requirements are biased accordingly. These requirements are summarized below:

i) In order for the system to sustain video gaming traffic generated by 450+ users and also to be future proof, the network should be high-capacity. In that, it should be able to adapt to future traffic demands.

ii) The system should be isolated from all other communication networks on the aircraft. Security of on-board communication infrastructure (i.e., flight critical networks) must not be compromised in any way, and any such possibility is eliminated by keeping the gaming network completely in isolation as a standalone system. In the proposed architecture, data is generated by the server and the clients for local consumption therefore no inter-connection with any other network is required.

iii) Since such new entertainment systems will be retrofitted and integrated with existing and future aircrafts, these have to meet some essential structural requirements. Flight safety requires that materials and weight of modules and devices used inside the aircraft must comply with national safety standards and therefore should be kept to the minimum. Indeed, an optical network is a very light-weight system due to the nature of fiber-optic cables, connectors, splitters and patch-cords. Active devices such as optical line terminal (OLT) and optical network terminal (ONT) are not bulky and are much lighter boxes than their electrical counterparts. Optical fiber cables and connectors weigh much less than the copper cables and connectors. Moreover, optical fiber networks are now being used in aircrafts for all other communication needs including sensing applications and therefore are not new to the industry.

iv) Electrical Power in aircraft is a limited commodity and needs to be utilized efficiently. Therefore, the proposed architecture must employ energy efficient infrastructure. Authors in [15-24] have proposed energy efficient protocols that save energy in optical networks. In a related work we have proposed observing traffic patterns to predict off-times, rather than reacting to idle times, for transceivers [24]. This results in considerable energy savings since it does not wait for the channel to become idle, rather predicts and takes action accordingly.

v) Network should be scalable, i.e., it should work with large aircrafts and as well as medium and small platforms. Gaming-Server needs to be able to cater for heavy and light traffic scenarios. The bandwidth allocation algorithm we proposed is dynamic, in that, it calculates key system parameters on run time depending on the number of users active in the system at that time. This is explained in detail in Chapter 4.

vi) Proposed network should be able to handle traffic for most types of games and even future enhancements. The flow of information should be independent of the application layer and therefore neutral. A modular approach is employed so that changes made at one layer do not interfere with the functionality of other layers as long as data is pushed up in the layered hierarchy in the correct format.

vii) Last but not the least, the network should be adaptable and cater to future enhancements that can be retrofitted using the existing fiber network. This is important to make sure that the system is future proof for the life of the aircraft (i.e., 20+ years). Quality of graphics is being improved in video game industry at a tremendous pace. High-resolution graphics essentially mean more data to be transmitted between console, terminals, and servers. The network design architecture should be able to handle increase in bandwidth requirements and future expansions.

Based on the above listed requirements, we designed and adapted the versatile and flexible architecture, flexible protocol stack, and bandwidth allocation algorithms that were simulated and analyzed for integrity and compatibility. These simulations were also measured against typical industry figures.

IFE systems have been present on-board airliners for a significant amount of time. The initial systems were able to stream songs only and after up gradation years later were able to stream videos as well. During the progression from audio to video, infrastructure was also upgraded to support basic videos using very low resolution screens and systems. It is well known however that IFE systems even in today's airlines do not represent the best of technology in terms of quality, speed and robustness. Companies such as Samsung, Thales, and Panasonic have developed systems that provide in-flight connectivity to the internet [10-12] from the aircraft cabin. However, these systems do not support high-speed connectivity for high-resolution gaming experience. Samsung and Thales are the leaders in developing seatback terminals. Focus of these systems is to ensure connectivity through email and social media applications. The most critical piece however is still missing from the picture. That piece is a well thought out transport system that can support all passengers (even for modern high-capacity aircrafts) with high-speed connectivity for a number of services including video gaming.

Depending on the type of video game, the game engine has to perform tasks and generate game play. Game play is what the player sees on the screen. It includes objects, lightening, self and other players, arms, ammunition, structures such as boxes and ramps etc. In the case of client side rendering, the server is generates motion vectors based on input from all clients present in the game (a change in position of one user must be shown to all users). Accordingly, the server has to inform each client about the changes happening in the gaming environment. For instance, if a player destroys a tank, all other players must be informed that the tank, as a structure, does not exist in the environment. Providing information about changes in the environment presents a critical time challenge. If transmitted too late, the information may be useless. If transmitted early it could give undue advantage to a player. Therefore, latency is one of the most critical parameters in video game system design.

1.3 Service oriented architectures

The concept of service oriented architecture is nascent. To understand what service oriented architecture is, let us take the example of local area network (LAN) which is a general purpose network. It can support a large number of services (i.e., Email/Exchange, Voice over IP (VoIP), audio and video streaming, video conferencing and streaming etc) with reasonable performance. It is not optimized for a particular service though; therefore it will perform nominally if it were to cater to only one service with high load. Performance depends on many factors that include type of service, tolerances in terms of data loss and delays and utilization of the network. While designing service oriented architectures all these parameters are optimized for a particular service, for instance, VoIP. This ensures maximum performance for the particular service although networks still work as a general resource for other less priority services if need be. Intelligent infrastructures are therefore considered the way forward. In later chapters, physical layer architecture, protocol stack and bandwidth allocation are introduced and discussed with the concept of service orientated architectures in perspective.

In the introductory chapter the statement of the problems has been presented with aims and objective. The needs and challenges have been briefly high-lighted. The current state-of-the-art and literature review has also been presented with some pertinent recent and original citations from the available resources. Rest of the dissertation has been organized in the following order: Chapter 2 presents the physical layer architecture and chapter 3 deals with the protocol stack. Next, the chapter 4 introduces bandwidth allocation. The chapter 5 concludes the dissertation with summarizing the design architecture and the simulation results and remarks regarding the validation of finding and the scope for future work. The references follow the chapter on concluding remarks and modeling code and software packages data and necessary info is included in the appendices, A through E.

CHAPTER 2: PHYSICAL LAYER

Optical networks once only being in the core now are increasingly playing an essential role in all parts of the network infrastructure in today's data driven world. Optical networks are inherently high-capacity, light weight, robust and greatly enable bandwidth hungry applications. Among the numerous ICT applications, these properties of optical networks render them very attractive for in-flight video gaming solutions and similar constraint but high-capacity applications. Modern video games require a lot more bandwidth, due to enhanced graphics, requiring ever increasing frame rates, and stability in the networks. Video gaming data attributes are discussed in chapter 3 with somewhat fine details. Various unique requirements for in-flight scenarios are that of the network needs to serve a large number of users, all of whom may be connected at the same time, as illustrated in Figure 1. Also, system should be able to adapt to changing number of active users. These and more requirements have already been discussed in chapter 1.

In the same context, in this chapter we propose standalone scalable and high-capacity network that can act as harbinger and enabler for real-time video content. Recently, several physical layer architectures have been proposed that enable high data rates in passive optical networks [5, 16, 25-30]. Improvements in data rates are realized by pushing the limits of technology and consistent with the information and communication theory [31]. Various modulation techniques are used to convert data into a transmittable signals' format. The signal that carries the data is called a carrier. Carrier signal in an optical network is light therefore data signal, which is comparatively at a low frequency, is modulated onto the optical carrier and transmitted over the optical fiber cable. Most basic form of modulation is on-off keying (OOK), which is a special case of amplitude shift keying (ASK) – which modulated the amplitude of the carrier signal. OOK refers to transmitting a voltage level when a binary “1” needs to be transmitted and keeping the laser diode off when “0” needs to be transmitted. It is intuitive to imagine that the speed of such a communication system is limited by how fast the laser diode can shift between the on and off states. Due to this bottleneck, complex modulation schemes have been developed and reported in literature that enables networks to carry more data on each of the active channels. Basic idea is to transmit two bits in one cycle of frequency instead of one bit per cycle. Such a technique effectively doubles the data rate. There are more complex techniques available today that can carry more than two bits per cycle. Two of the many modulation schemes that gained attention in industry are polarization mode quadrature phase shift keying (PM-QPSK) and coherent optical orthogonal frequency division multiplexing (CO-OFDM). QPSK, use 4 different phases of the carrier signal to transmit information. Hence, high data rates are achieved. This technique can be used in conjunction with using multiple polarizations of a signal. Data rate is calculated as $\mathrm{datarate} = \mathrm{symbolrate} * \mathrm{bits\ per\ symbol}$. On the contrary, OFDM modulates data onto low frequency carries which are later combined using inverse Fourier transform (IFFT) into a single time-domain signal which is modulated onto the optical carrier. There are definite advantages and dis-advantages of both the techniques and in our opinion OFDM holds great promise due to its flexibility. This is the reason why OFDM QAM was chosen over PM-QPSK, in this work. For completeness and quick-reference for the reader, OFDM is discussed in the next section.

In this chapter, we present a novel network architecture that is capable of providing approximately 195.2Mb/s (including FEC) data rate to each of the 450+ connected users. This downstream bandwidth is considered more than what is required at the moment for video gaming and would be able to cater for future gaming environments as well. Network is flexible and scalable therefore; it supports fewer users as well and in-fact bandwidth allocation increases for each individual user if fewer users are connected. This is discussed in much more detail in chapter 4 that deals with bandwidth allocation techniques. Aggregate data is transported using 16QAM OFDM signaling. Various dispersion compensations are required for long-haul and ultra-long-haul links with data rates of 10Gb/s and certainly for those that go beyond 100Gb/s [32], [33]. In our application domain electronic dispersion compensation would not be required because link lengths are quite small, and they are indeed almost negligible.

2.1 Physical layer technologies

In various domains of networks, data rates are largely limited by the physical layer technologies in place. Depending on the applications, different physical layer implementations can be adopted. Since the video-gaming involves an extremely data intensive operation for a network, we look for and adopt the architectures that offer higher data rates. Among various options OFDM is one of them that offer benefits that are discussed in the next section.

2.1.1 Overview of OFDM

OFDM has been used in other network technologies however, it's feasibility in optical networking has recently emerged due to its ability to increase the effective data rate in the network. It uses Fourier transform [34] to convert a set of low frequency electrical signals, shown in Figure 2, into a single time-domain signal that is modulated onto the optical carrier. Since low frequency signals are modulated separately with data, the technique is modular, in that; a sub-carrier (one of the low frequency modulated signals for one set of data stream) can be allocated to a particular user. This way the networks become contention free and therefore can work like a wavelength division multiplexed (WDM) system in principle although the technicalities do differ. One can also modulate each sub-carrier separately and perform time division multiplexing onto each one of them therefore serving more than one user with a single low frequency carrier. Lately, many techniques have been proposed and demonstrated for OFDM based architectures and curious reader should consult [16, 25, 28, 30, 35-37] for detailed reviews and latest proposed techniques.

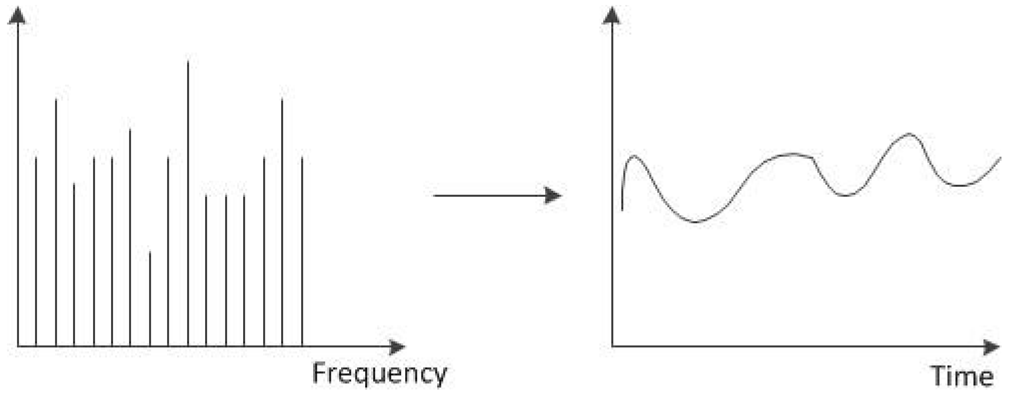


Figure 2: OFDM sub-carriers (frequency domain) to time-domain signal

Compared to other methods, an OFDM technique has certain distinct benefits. Some of these are as follows:

i) In OFDM, sub-carriers are orthogonal to each other therefore there is no chance of cross-talk between channels.

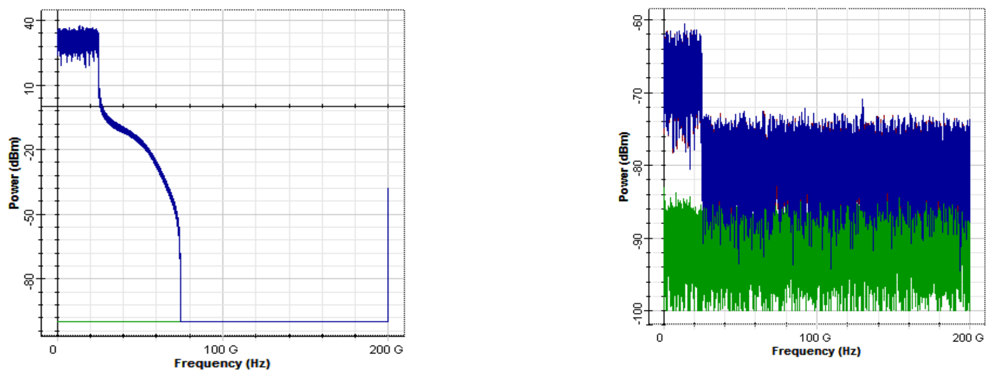
ii) An OFDM symbol is made up of bits, one from each sub-carrier data streams. Therefore if a symbol gets distorted not all data is corrupted. This adds to data integrity in case of poor channel conditions. Symbol rate is lower than the data rate therefore guard-band can be used between symbol to reduce inter-symbol interference (ISI).

iii) OFDM gives system designer an opportunity to utilize full available spectrum and thus to improve spectral efficiency.

iv) OFDM allows the system to allocate a separate channel to every user which ensures privacy and security of data channel. Also since crosstalk is not present, the data is not corrupted.

v) An OFDM signal can handle filtering better because time domain signal is not distorted at the decision time. Passing an OFDM signals for instance, through a low pass filter does not change the signal at the decision time, therefore signal can be interpreted correctly.

An ideal OFDM signal, in frequency domain would look like the one shown in Fig 3a below which happens to be from one of the simulated results at the source. Figure 3b shows an OFDM signal that has travelled a distance. Due to the fast processing high-frequency components are added to the signal which act as noise and must be removed before interpreting the signal. From Fourier synthesis one knows that a large number of sine waves are required for constructing a perfect square wave signal illustrated in Fig 4. Sinusoids used to achieve this are periodic harmonics of the fundamental frequency (i.e., 1/3rd, 1/5th, 1/7th and so on) also evident from Figure 4. FFT follows from the fact that any time domain signal can be created if the right amount of sine waves are used each at the a certain frequency. Therefore, OFDM can be spectrally very efficient, however, complexity of calculations increases drastically with an increase in the number of sub-carriers. Therefore a balance has to be reached for practical scenarios. The balance is reached by rigorous simulations with carefully configuring the parameters.



(a) (b)

Figure 3: OFDM signal (a) OFDM signal in frequency domain (b) Non-filtered OFDM signal in frequency domain after travelling a distance

Figure 3 above shows the results from one of our simulated environments detailed later in this chapter.

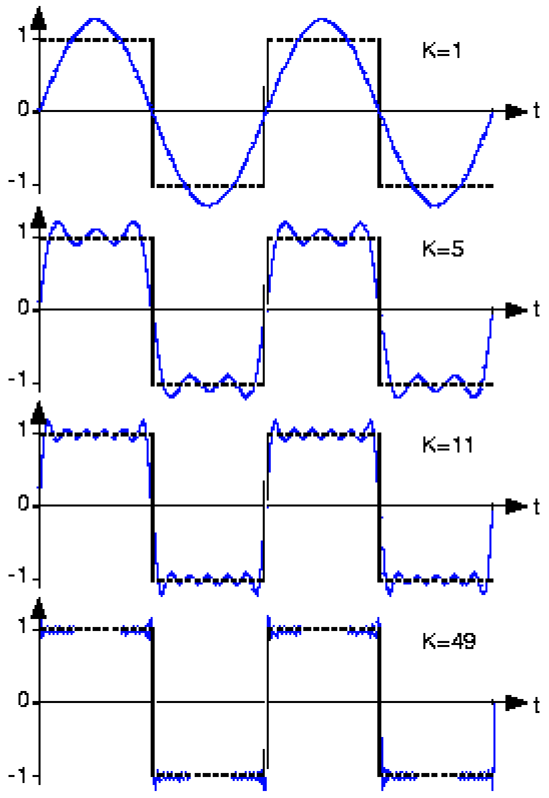


Figure 4: A square wave generated with multiple sine waves [38]

2.1.2 Overview of TDM / Single channel

Time division multiplexing (TDM) is a seasoned technique for allocation of time in communication systems [39]. TDM can be modeled to work in a manner that it not only improves system performance but also reduces the latency. TDM based schemes allocate time on a shared channel to contending users. In conventional TDM (also known as statistical multiplexing), users are allocated time irrespective of their need or demand. This means that a user is allocated time slots in the absence of any demand and need and thus the allocated bandwidth gets wasted. A simple TDM based allocation scheme for $N = 22$ users is shown in Figure 5. It is readily evident

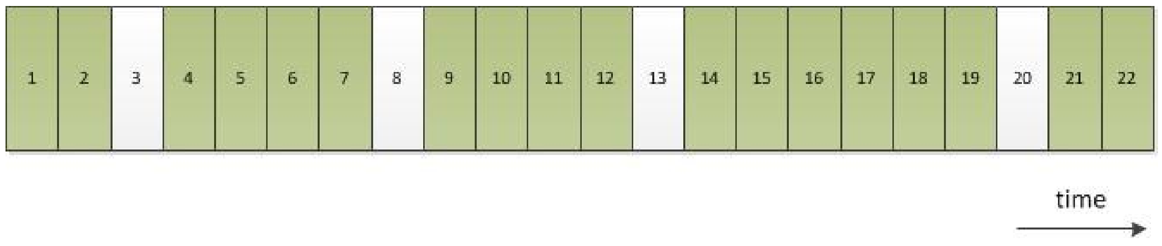


Figure 5: Typical time division multiplexing (TDM) allocation cycle

that although users 3, 8, 13 and 20 do not have any data to transmit, yet they still get time to transmit. Such deficiencies have been improved a lot in TDM based systems and in chapter 4 we show that TDM can still be used to allocate bandwidth for a large number of users as well.

The next section introduces the proposed physical layer architecture for high-capacity high-fan-out applications.

2.2 Proposed architecture

For demonstration of feasibility of OFDM, we used 16QAM, which provides relatively increased noise immunity with a reasonably large decision dynamic range for the receivers. Figure 6 below shows the segment of network architectural layout. Dotted box section on the top-left shows the transmitter and dotted box section on the bottom-left shows the receiver with distribution sub-system shown to the right. For full fan-out, signal has to pass through two splitters on each link. A 1 X 16 splitter in tandem with a 1 X 32 splitter enable a fan-out of 450+.

2.2.1 Details of Network Segment used in Simulations

This section details the design parameters and configurations for the proposed architecture.

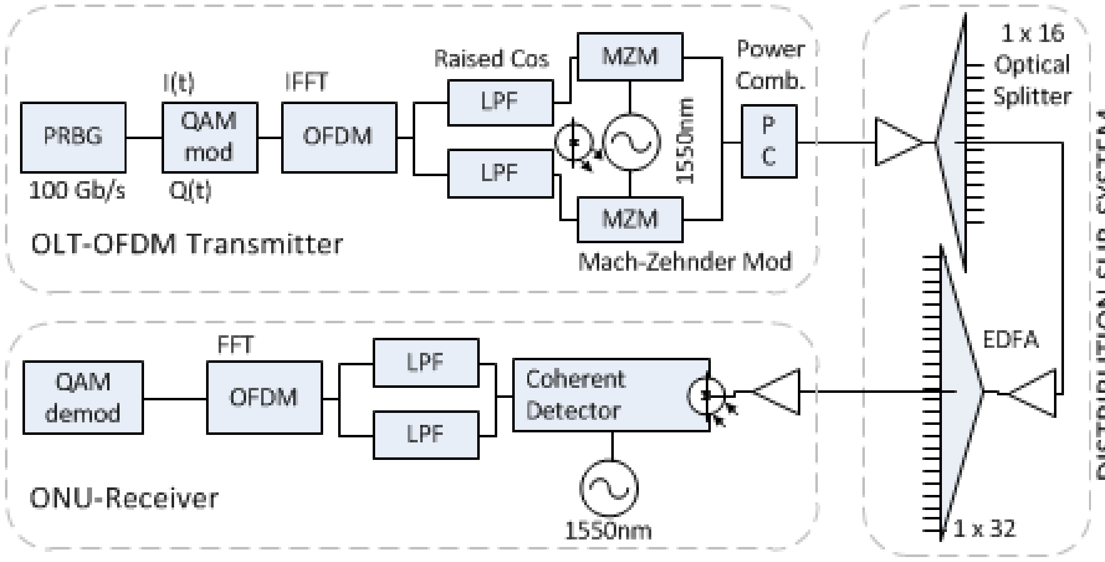


Figure 6: Block Diagram of proposed architecture. Three major components are the OFDM transmitter, distribution system and the OFDM receiver

The illustrated diagram shows the architecture of the OFDM based architecture. Transmitter and receiver components are discussed.

a) Transmitter: Pseudo Random Bit Generator (PRBG) generates random bit sequence at the rate of 100Gb/s. The PRBG mimics our aggregate message signal that needs to be transmitted. QAM generates 4-bit symbols and splits this signal into two orthogonal components i.e., $\mathrm{S(t)} = \mathrm{A_I}\ \mathrm{(t)Cos(wt)} - \mathrm{A_Q}\ \mathrm{(t)Sin(wt)}$, where $\mathrm{A_I}$ and $\mathrm{A_Q}$ are amplitudes of the real and imaginary (orthogonal) part of the signal and w is the frequency. Since each symbol represents 4 bits, this is a 16-QAM system i.e., $\mathrm{S} = (0001,0010,0011, \dots ,1111)$ with baud-rate of 25GBaud/s. On the constellation each point represents a symbol with its unique amplitude and phase. At the receiver, we expect this diagram to be as less distorted as possible for data to be extracted successfully. OFDM converts output signals (I and Q) from the quadrature amplitude modulator (QAM) to a composite time domain signal. I-phase is the in-phase signal at the output of the QAM and Q-phase is the quadrature phase signal at the output of the QAM.

Large number of OFDM sub-carriers help with the applications requiring scalability. Since sub-carriers can be individually allocated, although large number of sub-carriers also has its own limitations that must be kept in mind. Close packing of sub-carriers results in cross-talk. Inverse Fast Fourier Transformation (IFFT) and Fast Fourier Transformation (FFT) at the transmitter and receiver respectively, perform multiplications with a complexity that increases linearly with the number of sub-carriers therefore overhead is minimum [40]. We use $\mathrm{N_{SC}} = 1024$ sub-carriers. OFDM symbol period is dependent on QAM baud-rate and number of sub-carriers used. Therefore, $\mathrm{T_{OFDM}} = \mathrm{N_{SC}} * \mathrm{T_{QAM}} - \mathrm{BAUD}$. Also, $\mathrm{baudrate} = \mathrm{bitrate/QAM}$ bits per symbol. As an example, output of one of the OFDM signals (I-phase) is shown in Figure 8b. Q-phase (quadrature phase) signal looks similar but is out of phase. A raised cosine filter, low-pass-filter (LPF) is used to filter high frequencies that are generated as a result of fast variations brought about in the carrier signals during modulation. Using this filter allows us to satisfy Nyquist criteria for inter-symbol interference Time domain representation of the signal changes due to filtering but only at instances in time that are not the decision times. Data rate of 100Gb/s is sent over 25GBaud/s in a 16QAM system. Spectral efficiency can be calculated to be 4 bits/sec/Hz. 1024 subcarriers travel inside one OFDM symbol therefore OFDM symbol rate is $\mathrm{R_{OFDM}} = \mathrm{baud\ rate/NSC}$ =24.4MSym/sec. Each subcarrier carries 97.6Mb/s and each user is allocated 2 subcarriers amounting to 195.2Mb/s, ideally. This translates to a 2.56THz modulated signal centered at 193.4THz. Nyquist bandwidth is 12.8THz, when symbol duration (TS) is 0.04ns. Filtered signal is shown in Figure 8c. I-phase and Q-phase are electrical signals at this point and must be modulated onto optical carriers. A CW optical carrier centered at 1550nm (193.4THz) is used as carrier signal. LiNbO3 based Mach-Zhender modulators (MZM) are used to modulate the OFDM signal onto the optical carrier. Both I-phase and Q-phase signals are modulated separately and combined using an optical combiner and the combined signal, as shown in Figure 8e, is transmitted.

Passive optical splitters are used to split optical signal for distribution. Single stage Erbium-doped fiber amplifiers (EDFA) are used to amplify the signal. Since splitters are employed in tandem, amplifiers help keep power level high enough for reliable detection. However, recurring amplification adds noise to the signals.

b) Receiver: At the receiver end, optical signal is distorted due to various impairments and noise. One of the reasons for using a PIN photo detector is its ability to detect low power signals due to high sensitivity and responsivity.

Optical signals are passed through the coherent optical receivers that separate two orthogonal signals using four (4) PIN-photodiodes connected in balanced configuration. Received signal and local oscillator (LO) signals are coupled using 3dB couplers. One component of LO is shifted by $\frac{\pi}{2}, \eta A e^{-j(\theta-\varphi)}$, where $\varphi = \pi/2$, using phase shifter. The other components (non-shifted) coupled with the incoming signal as they are. Advantage of using a coherent receiver instead of an optical filter to isolate the frequency of choice is that coherent receiver allows more precise detection with strong signal from the weak incoming signals. For example, down-converted signal is shown in Figure 9c containing high frequency components. The low pass filter (LPF) stops the high frequency components to pass through to the OFDM receiver, as illustrated in Figure 9d. These two signals are the I-phase and Q-phase signals that are fed into the OFDM receiver for FFT to convert serial bit streams into its frequency components. Resultant signals are fed into the QAM decoder that generates the bit stream by detecting respective symbols, illustrated in Figure 9e. Distortion in the received signals arises due to slight changes in phase of the signals and variations in amplitudes.

2.3 Experiments

This section details the components used in simulations and their configuration. A component level view of the architecture is illustrated in Figure 7. The design and simulation of the architectures was carried out keeping in mind the requirements listed in chapter 1 i.e., high-fan-out. Distortion in amplitude or phase of the modulated signal results in a distorted constellation diagram at the receiver and renders the decision-making more difficult and in some cases impossible. Since phase jitter shifts dots on the constellation from one phase to another creating an arc like representation, it is hard to decode. Similarly, with changes in amplitude constellation diagram converges to the center (mostly) and makes it hard to decode. A combination of both of these occurrences

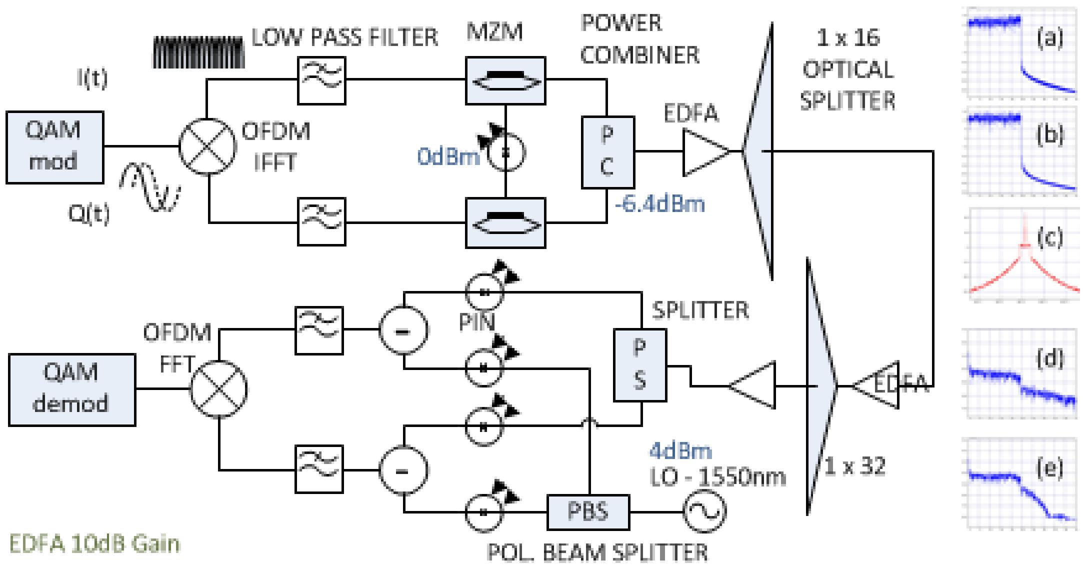


Figure 7: Component level view of proposed architecture. Transmitter: (a) OFDM signal generated after IFFT using NSC subcarriers, non-filtered, 25GHz (b) Filtered output of OFDM to be modulated onto the optical carrier at 1550nm. Roll-off of 0.5 and cut-off set at 0.62. Link: (c) Transmitted optical signal with modulated OFDM signal. Receiver results (d) down converted in-Phase signal at the receiver (e) signal after passing through raised cosine low pass filter. BER of 5.4 * 10-3 (Figure 10).

gives us a spiral-like constellation illustrated in Figure 8f. We also get similar constellation diagrams at the receiver when there is a discrepancy between number of encoded and decoded sub-carriers.

2.3.1 Simulation software

Optsim^{TM} was used for simulations of various network layouts and architectures. Optsim^{TM} is an industry leading optical network simulation software suit from RSoft^{TM}. The latest build-out allowed us to simulate the architectures based on coherent detection with more realistic physical parameters. Coherent detection though studied earlier is relatively new in the practical optical networks domain compared to direct detection technology. However it is considered to be one of the most promising ways to go forward owing to its advantages over other conventional techniques in the high-data rates of

20Gb/s and beyond. Indeed for 40Gb/s and higher data rates there are hardly any other techniques in the literature that can address the physical limitations.

2.3.2 Results

This section introduces and presents the results based on the simulations studies and investigations of Coherent OFDM QAM based architectures. For the demonstration purpose, initially, 2 bits per QAM symbol were used with 512 OFDM sub-carriers. Figure 8 a – e, shows the signal as is proceeds through the transmitter. Figure 9 illustrates the receiver. One can see that the QAM signal, represented by the 4 unique symbols shown in Figure 8a. This gives us a bit rate of 2 bits per symbol. Figure 8b illustrates the

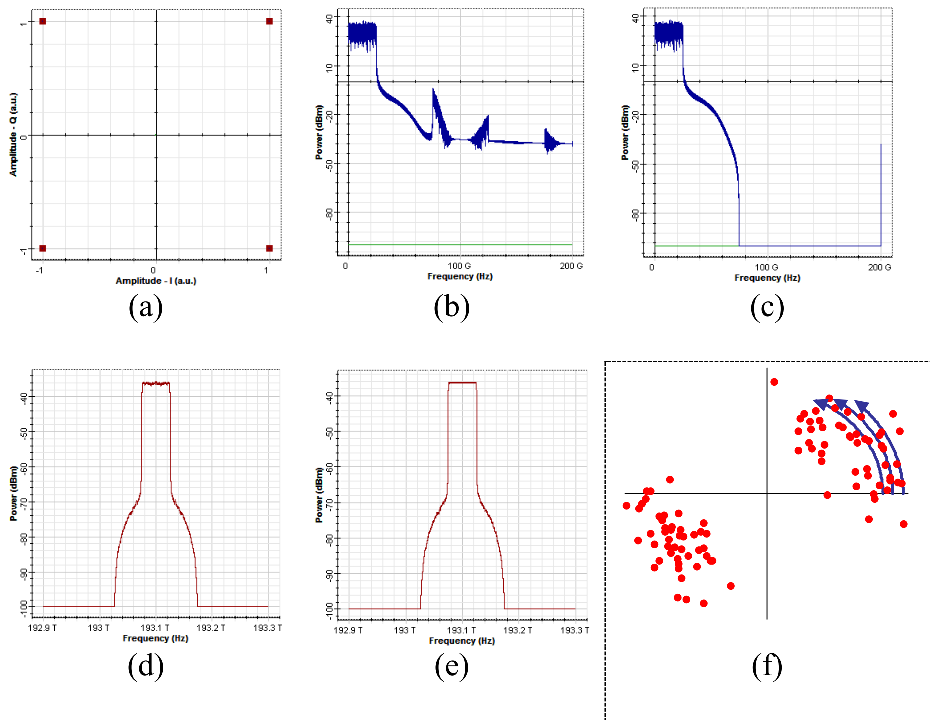
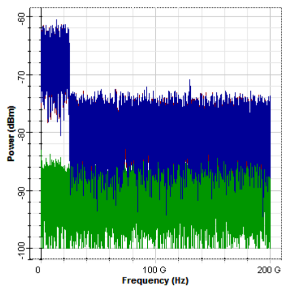


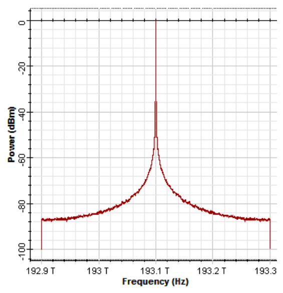
Figure 8: Coherent 100Gb/s transmitter with 2bits/symbol. (a) Constellation Diagram QAM Sequence Generator generates 2 sequences (b) OFDM Modulator (c) LP Filter (d) Output from MZM (e) Power combiner Output from both MZM combined for transmission (f) distorted constellation diagram for a 2 symbol system (just for illustration, not connected with Figures 8a – 8e)

OFDM signal in the frequency domain followed by filtered signal in Figure 8c. It should be noted that this is one of the two orthogonal signals modulated separately using Mach-Zehnder modulators output of which is shown in Figures 8d and 8e.

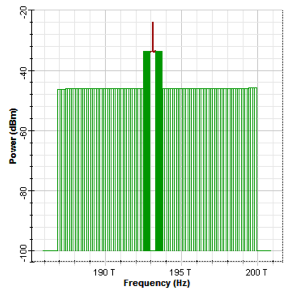
The receiver related signal sequence is depicted in Figure 9 below.



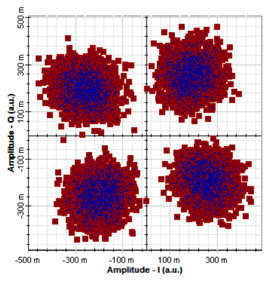
(c)



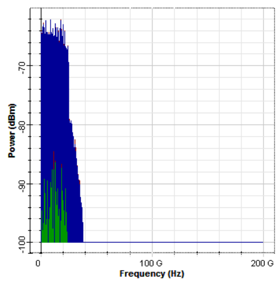
(b)



(a)



(e)



(d)

Figure 9: 100Gb/s receiver (a) Incoming signal spectrum (b) Local Oscillator 193.1 THz - 2dBm (c) Limiting Amplifier (d) bandwidth limited signal (e) Received Optical Signal RED = Signal BLUE = noise

Figure 9a shows the received signal. Local oscillator's output is illustrated in Figure 9b, followed by the decoded OFDM signal in Figure 9c. Since noise is visible in the form of high frequencies, a LPF is used to filter out high frequency components, illustrated by Figure 9d. Figure 9e shows the reconstructed QAM signal with reasonable noise that is to be expected.

Figure 10 shows results achieved from other simulations with more complex signal encoding such as 16-QAM with 4 bits per symbol. For the following BER discussions we use the results achieved by the 16QAM simulations. Received signal BER is in agreement with [35] and FEC threshold. It can be seen in Figure 10 that increases in number of subcarriers effects BER performance. Similarly, received power affects the BER performance as shown in Figure 11.

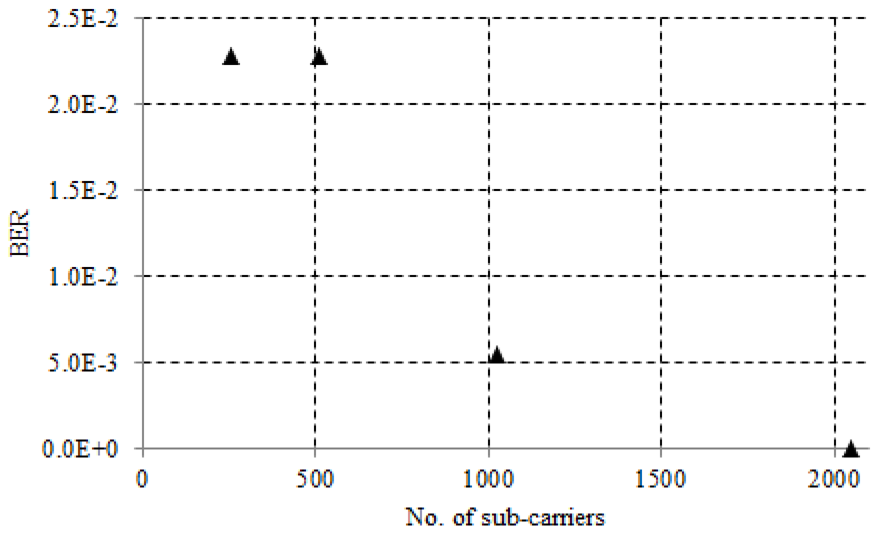


Figure 10: Effect of increasing the number of subcarriers on BER. Increasing number of sub-carriers increases bit error rate significantly but shows unpredictable dependence.

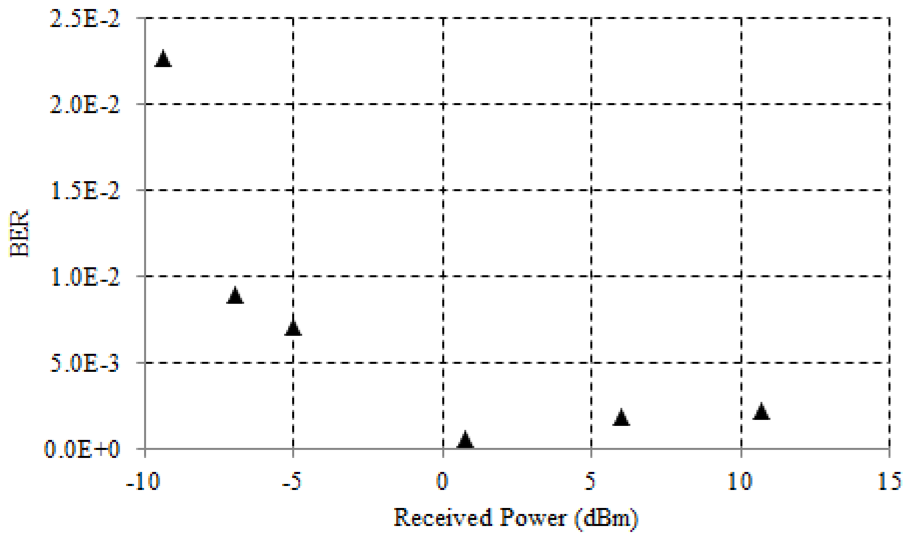


Figure 11: Received signal power and its effect on BER performance for the 16-QAM OFDM signal. With two amplifiers connected in tandem between the splitters we get reasonable BER.

It is evident that the threshold BER performance varies with the number of sub-carriers used as well as with the power received by the receiver, which dictates using the optimum number of sub-carriers for a desired configuration.

2.4 Discussions and relevant remarks

Two architectures variations were discussed in this chapter with results pertaining to proposed 16QAM OFDM based coherent architecture. We showed that in a high-fan-out optical network 450+ simultaneous users can be supported with reasonable signal performance within the typical limits. The physical layer infrastructure helps us develop upper layers for optimizing full system performance. Proposed and investigated architecture is best suited for high-capacity bandwidth hungry applications such as in-flight video gaming systems. Proposed architecture will be able to provide up to approximately 200Mb/s to each user with BER kept under the threshold of FEC using fair allocation.

In the next chapter (3), we discuss the protocol stack and data/datasets used for developing models for upper layers such as bandwidth allocation in chapter 4.

CHAPTER 3: PROTOCOL STACK

As introduced in previous chapters, the high-fan-out optical network is the workhorse of the scalable high-capacity optical networks for constrained environments. At the physical interface, data is collected between the optical network terminal (ONT) and optical line terminal (OLT) as a variable bit stream. These bits form data packets that have headers with information about source and destination routing. When a packet arrives at the network interface (also known as the Network Interface Card or NIC) it is passed on as "layer by layer". A set of layers that a packet must traverse before being transmitted at the sender and before being read at the receiver constitutes a protocol stack. Stack refers to the protocol layers virtually sitting on top of each other as shown in Fig 12.

In the following sections we contemplate the protocols in place for data transport and define real-time video-game stream (RVGS) protocol envisioned for transportation of video game data across the network.

3.1 Initial considerations

Each layer in the protocol stack associates and attaches information to the data is receives from the upper later, as depicted in Figure 12 below. Associated information carries specific functions that are necessary for transmission to be completed from one end to the other.

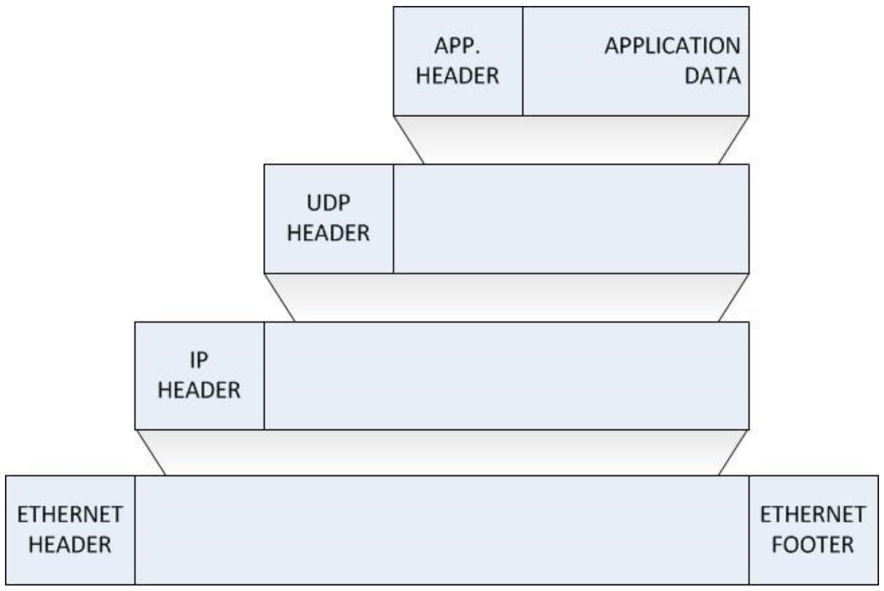


Figure 12: Headers information added by each protocol layer

In Figure 12, application data refers to the data generated by the video game server in both CSR and SSR based systems. Application attaches information about data that it needs to transport such as "data length" in bytes. This is followed by user datagram protocol (UDP) header that contains port numbers (source port and destination port) that uniquely identify the application in the app header. As the packet arrives at the computer port number identifies the application the packet belongs to. Subsequently, the IP header attaches the course and destination IP addresses to the payload. These addresses help in routing the packet all the way from source to destination. Since UDP is a connection less protocol, each datagram is routed separately as a unique entity through the network. This also means that UDP packets may not arrive at the receiver in order and need to be ordered by the application. An alternative, transmission control protocol (TCP) is not suitable for such applications due to the delays it introduces into the data streams. TCP handles contention which increase delays on top of its heavy header therefore TCP is not considered a good choice for video gaming data transport. IP header also contains other information such as flags, time-to-live, fragmentation bit and IP version number. In the end, and just before transmission, the IP packet is encapsulated inside an Ethernet packet also known as a Frame, as illustrated in Figure 13 as well. Successive frames are transmitted over the EPON in order to complete a transmission.

Since the receiver also has all the respective layers, a packet that arrives at the receiver goes through the reverse process at each layer. Each layer removes the header and passes on the packet to the upper layer until data is finally transmitted to the application.

This process is similar to Amazon^{TM} sending a shipment in its own boxes although products are already packaged by the manufacturers. Since Amazon is only a transporter, it is concerned only with end-to-end delivery and not with the contents of the package therefore a plain brown box with shipping labels is enough for the transportation. The manufacturer however puts the product in its own packaging with labels, trademark, and the product information, not for Amazon, but for the end consumer. For a consumer, the specific information such as warranty and specifications are highly important. Amazon analogy is also quite suitable because they have streamlined their processes to ensure a fast, safe and accurate delivery. This is much like the service oriented architecture discussed in earlier chapter 1. Continuing Amazon analogy, for an overnight delivery Amazon uses FedEx's premium network while a 3-5 days delivery is sent over USP's network. Both types of networks are optimized for certain specialty services.

Any communication would work in a similar fashion on the most part. The differences lie primarily in stack implementation i.e., the type of protocol being used by a particular service.

It should be noted that header information is considered an “over-head” that must be transmitted in order to correctly and securely transmit information over a public network. Also, the data to be transmitted is almost always much larger than the capacity allowed packet size therefore data is fragmented and divided into smaller chunks and inserted into packets one after the other in a sequence.

Packet sizes are defined for almost all networks dedicated to support specific services. Consequently, depending on the specific network architectures packet sizes may vary. These are called maximum transmission units (MTU) [41]. Table 1 below gives view of packet sizes in a UDP/IP stack.

Figure 13 shows an Ethernet packet that carries real-time video game stream (RVGS). This can be a SSR video frame or CSR based position vectors’ information for rendering at the client. From the Ethernet packets’ perspective, it is simply data or payload (recall the Amazon analogy).

TABLE 1
Packet size UDP/IP stack

Protocol	Payload size (bytes)	Packet Size (bytes)
Ethernet frame	1500	3 (header) + IP packet size
IP	PDU size	20 (header) + PDU size
UDP	RVGS size	8 (header) + RVGS size

Real-time video game stream (RVGS) is the protocol that carries video game data for both SSR and CSR streams. RVGS is discussed in detail in the next section. PDU refers the protocol datagram unit which is a general term used for a packet for a particular accompanying protocol type.

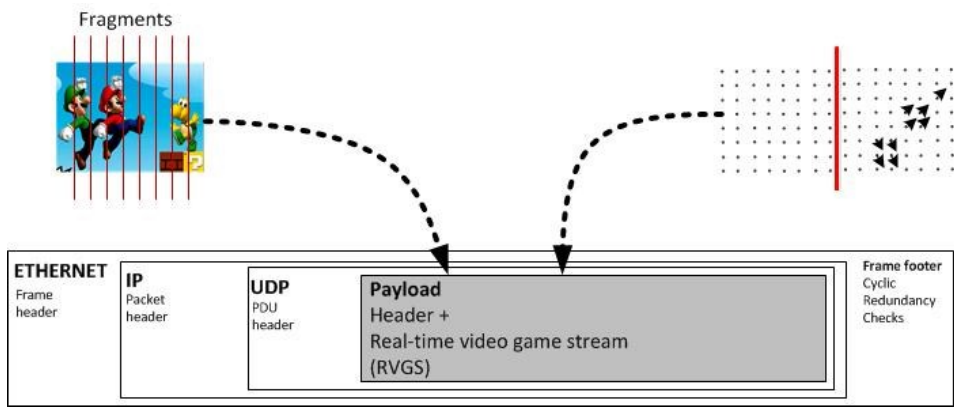


Figure 13: UDP/IP protocol stack on Ethernet channel

Maximum size of an Ethernet packet is 1518 bytes or 12144 bits [42]. This includes data and all the headers that must be placed in order to make ready the packet for transmission. Therefore, maximum payload an Ethernet packet can take is 1500 bytes. Data enters the system from the physical interface and makes its way to the top layer (application layer) after passing through the UDP/IP stack [43].

In a SSR system, software is installed at the client end within with the game play is shown. The software connects with the remote video game server and essentially streams video game frames in the downstream and the user interactions in the upstream. Depending on the resolution and frame rate (frames per second, fps), we should expect to see a much larger data rate compared to CSR system.

On the other hand, in a CSR system, vector information is transmitted as payload in the downstream. This means more information related to game play is transmitted within the same span of time. This behavior is discussed in detail in the next section followed by deduced bandwidth requirements from the datasets.

Cycle times are optimized to equal packet lengths. For a low weight user, this means at least one packet will be transmitted in every cycle. For a high weight user, if there is enough data to be transmitted, at least three packets can be transmitted. Priority weights and assignment mechanism is discussed in great detail in chapter 4.

3.2 Real-time Video Game Stream (RVGS)

RVGS is a novel UDP/IP stack compatible transmission protocol that specifies the format in which video game data is transmitted over the network. Figure 14 illustrates RVGS packet header. Service specific protocols such as file transfer protocol (FTP), simple mail transfer protocol (SMTP) and real-time transport protocol (RTP) have been in use for a long time in communication systems however a protocol specific to video game traffic has not been proposed until recently, despite tremendous growth in the video game industry. To the best of our knowledge there is no other current work related to video game traffic protocols available for reference except for one proposed recently in [44] and a protocol that was presented back in 2002 [45]. PGTP proposed in [44] deals with the video game traffic targeted towards smartphones for mobile game play. Authors propose an energy efficient protocol to save battery power while playing a video game on a smartphone. It is shown that delays are kept under typical tolerances. Although it can transport video game data, just like UDP or any other protocol can, it cannot perform equally good for both types of video game stream i.e., SSR and CSR, due to lack of this segregation of traffic in the paper. Authors in [45] propose game transport protocol (GTP) that deals with online event driven data transport for video games for older video games utilizing windows to control the flow of information. GTP is a TCP based protocol

When compared with PGTP and GTP, RVGS is inherently targeted towards video game data in terms of inner video game details, such as information about type of data stream (i.e., SSR or CSR), being transmitted in the header. Since RVGS is video game specific there is more control on the information that is transmitted. This makes RVGS more of an application layer service oriented protocol which is lightweight due to its ability to hold more meaningful information in the header.

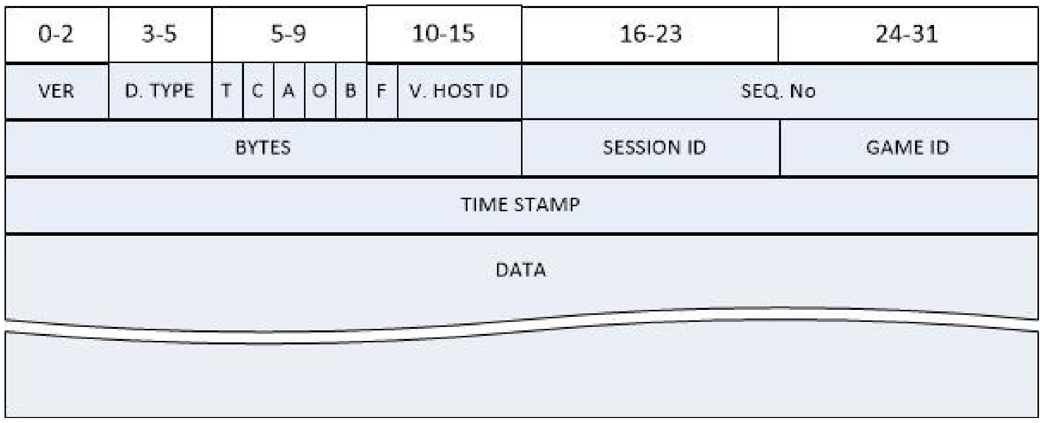


Figure 14: Real-time Video Game Stream (RVGS) packet header and data

The details about the fields used in RVGS packet header are as follows:

VER (3 bits): Provides information about the specific version of RVGS protocol header used for transmission.

D. TYPE (3 bits): Describes the type of data that has arrived inside the payload/data portion in the RVGS packet i.e., Audio data, video data, environment variables, player info, admin info, settings. Note that some of the data may need acknowledgment.

T (bit): Frames/Vectors

C (bit): Check data bit is used to put specific checks for the data that is encapsulated in the data payload portion of the packet. This field may be unset so that no checks are performed and UDP/IP generic checks are used. This bit can be overloaded to perform other Boolean tasks as well.

A (bit): Generally acknowledgements are not used in real-time transmission due to delays incurred in transmitting and waiting for acknowledgments. However, there are types of data that may require acknowledgment. Such acknowledgments do not interfere with game play therefore can be set to acknowledge data like settings and administrative controls.

O (bit): If packet contains completely non-conventional data, other data bit can be set to caution the receiver not to decode it as a video game stream.

B (bit): Blank bit is set when there is not data present besides the header and the packet acts as acknowledgement with the sequence number field with the acknowledgment number.

F (bit): Fragment bit is set when the information in the payload is contains a fragment of the whole, for instance, a fragment of the entire frame.

V. HOST ID (5 bits): Multiple game sessions may be active on a host. Virtual Host application ID identifies the particular game session.

SEQ. No. (16 bits): UDP packets are not ordered due to lack of connection establishment at the transport therefore sequence numbers help in ordering the packets at the receiver. Sequence number reset after the maximum is reached.

BYTES (16 bits): Stores the number of bytes in the packet including payload.

SESSION ID (8 bits): Session the packet belongs to

GAME ID (8 bits): Game ID on the host (for multiple games) TIME STAMP: Time stamp synchronization

DATA: Payload/data

As illustrated in Figure 14 the data is variable. This is suitable for both CSR and SSR streams. Same amount of information about the game play will take many more packets in SSR stream compared to CSR stream which will only transmit change vectors and audio information. BYTES field helps in keeping track of number of bytes in the packet.

As mentioned earlier, RVGS is compatible with UDP/IP protocol stack, as shown in Figure 13, and works seamlessly like other service specific protocols. The reason for designing a new protocol rather than using an existing one such as RTP is that RTP was designed for media related information, such as audio/video streams. The major difference between a video game stream and a regular media stream is the detail in information that is transmitted. Both the streams are inherently different in nature. For instance, audio/video streams are pre-coded data bits that are decoded at the receiver pretty much in the same manner for as long as the stream is relayed. A video game stream has much more detail in terms of change vectors and other player related information. A stream can be frames or vectors for example. Complexity of decoding at the receiver increases if this information is not formatted, more so, if the information is transmitted in a packet format that was developed for another type of service.

3.3 Data and observations

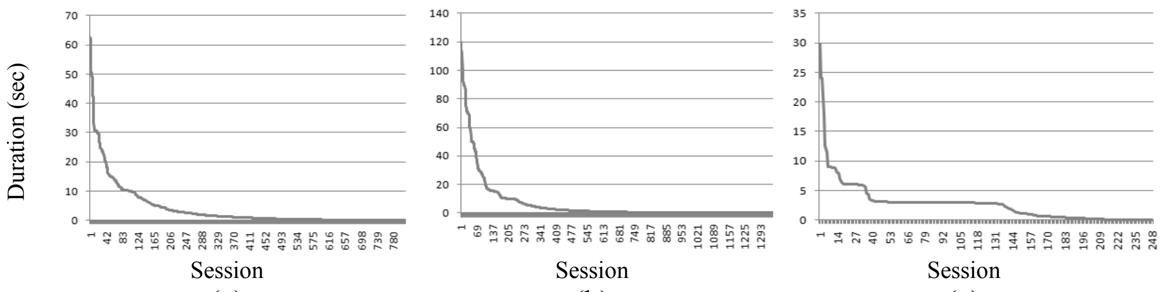
Analysis for bandwidth allocation in networks begins with identifying bandwidth requirements for particular services. Bandwidth requirements ought to be based on real-life data for realistic analysis. It is for the same reason that we use datasets gathered from various sources throughout the course of this dissertation project and these are discussed in detail in this chapter. From the network interface, data was captured using Wireshark^{TM} – a standard data capture tool used extensively in networks research. These were SSR and CSR based video game related datasets that were captured at the local interface during online gameplay from OnLive [1] and local game play (computer to computer, single and multi-player). Other network traffic datasets were downloaded through CAIDA [8]. These datasets represent internet traffic at the core. Details about these datasets have been listed in Table 2 for a quick reference and discussed below.

i) Datasets downloaded from CAIDA were captured at ultra-high-speed interfaces in Chicago and San Jose, therefore, in order to use them, local traffic information was extracted from them. Although not solely containing video-game data, these datasets provided and insight into the way internet traffic works. The tedious task of extracting local information required sniffing through large datasets to look for unique source and destination pairs, which would then be categorized as data streams originated from unique sources such as individual users or organizations. Packet sniffing for datasets was done using code written in C++ using standard "lpcap" libraries for capturing traffic from interfaces (Appendix E). After identifying communication sessions, session lengths, arrival and other parameters were observed.

Figure 15 shows the nature of internet traffic which predominantly comprises of transmission control protocol (TCP) packets. TCP is a connection oriented alternative protocol to UDP. It can be seen the data follows Pareto distribution very closely. Pareto distribution is a long tailed distribution that represents large number of smaller sessions and a small number of larger sessions [24]. Just for reference, Figure 16 shows snapshots of communication sessions (between unique source destination pairs) extracted from the same datasets. When plotted on a time graph it is easy to see that the sessions occur in bursts. It can also be observed that large number of smaller sessions and a small number of larger sessions are present in each communication, which follows Pareto distribution. Figure 16a shows a relatively large session at the data rate of OC-48 (Optical Carrier -48) (OC=51.84Mb/s) which equals about 2.5Gb/s (OC * 48) at the interface where it was collected. One can identify that the proportion of smaller sessions is larger than that of larger sessions. Figures 16(b – d) show sessions extracted from the high-speed data. Figures 16(e – f) are snapshots of higher data rate at OC- 192 (Optical Carrier-192) which is approximately 10Gb/s at the interface. Rest of Fig 16 shows the extracted sessions from the large dataset shown in Fig 16e. In all fairness, it is hard to say if there was any co-relation between the separate communications shown in Figure 16, but we can say with a high level of certainty that within a communication, all sessions were co-related since they were communicating between the same source-destination pairs.

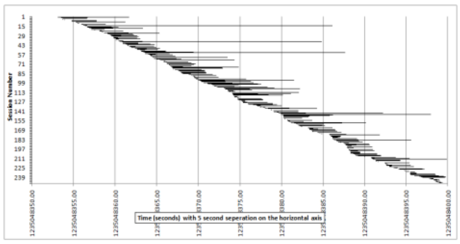
Many more datasets were also analyzed and were found to be following similar patterns however, they cannot be used for specifically video-game purposes due to lack of evidence about them carrying video-game data. This is the reason why latest video game datasets were collected and are discussed next.

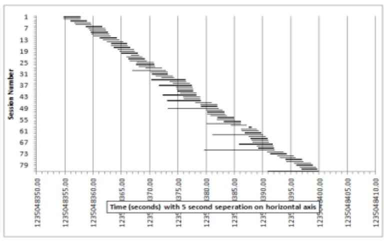
As far as the discussed datasets are concerned, conclusive evidence can be drawn from them for proposing energy efficient schemes, by virtue of having idle times, in optical networking, which was carried out as a separate work during the project and in reported in [24].

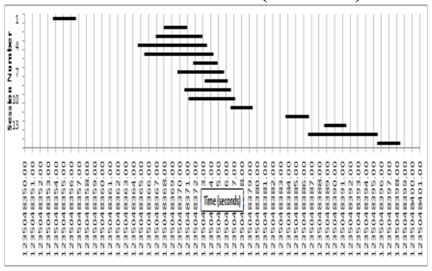


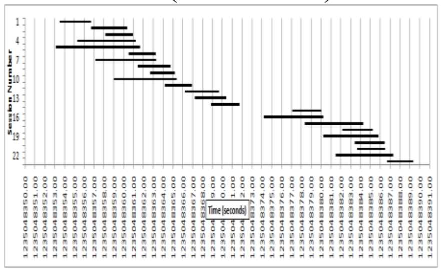
(a) (b) (c)

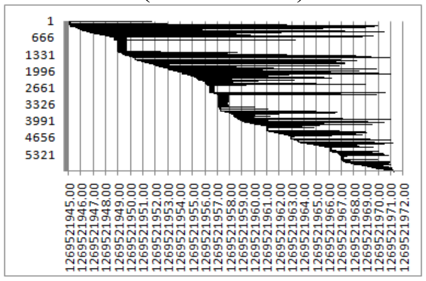
Figure 15: Data extracts from CAIDA datasets (a) Chicago OC-192 (b) San Jose OC-192 (c) Chicago OC-48

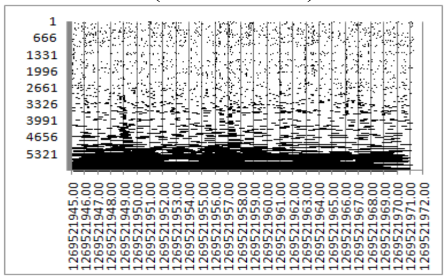

(a) Session number vs Arrival time and duration (OC-48)

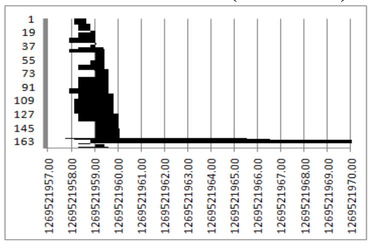

(b) Session duration vs Arrival time (Local LAN)

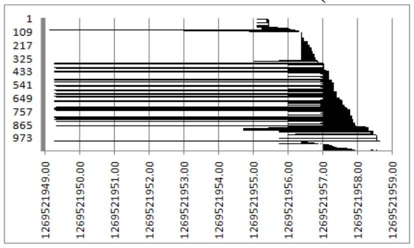

(c) Session duration vs Arrival time (Local LAN)


(d) Session duration vs Arrival time (Local LAN)


(e) Session number vs Arrival time and duration / (OC-192)

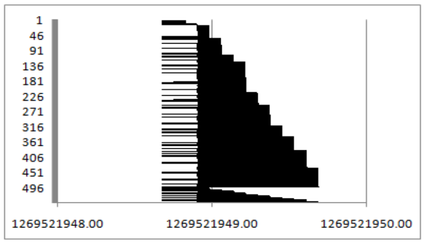

(f) Session number vs Arrival time / distribution of sessions (OC-192)

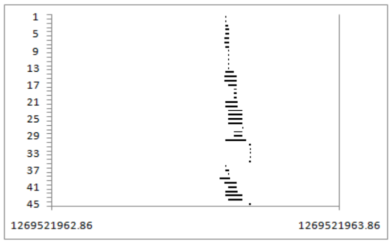




(g) Session number vs Arrival time (OC-192) / session spanning 13 seconds

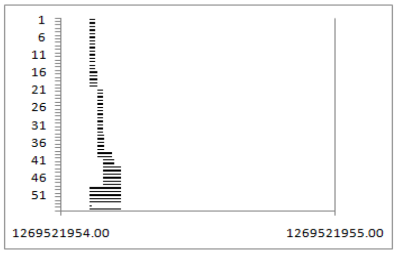
(h) Session number vs Arrival time (OC-192) / session spanning 11 seconds

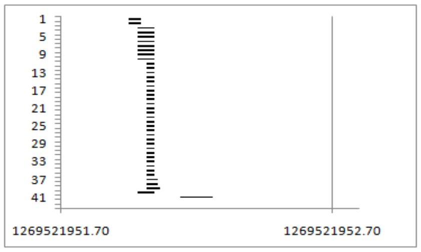




(i) Session number vs Arrival time (OC-192) / session duration < seconds

(j) Session number vs Arrival time (OC-192) / session duration < one second





(k) Session number vs Arrival time (OC-192) / session duration < one second

(l) Session number vs Arrival time (OC-192) / session duration < one second

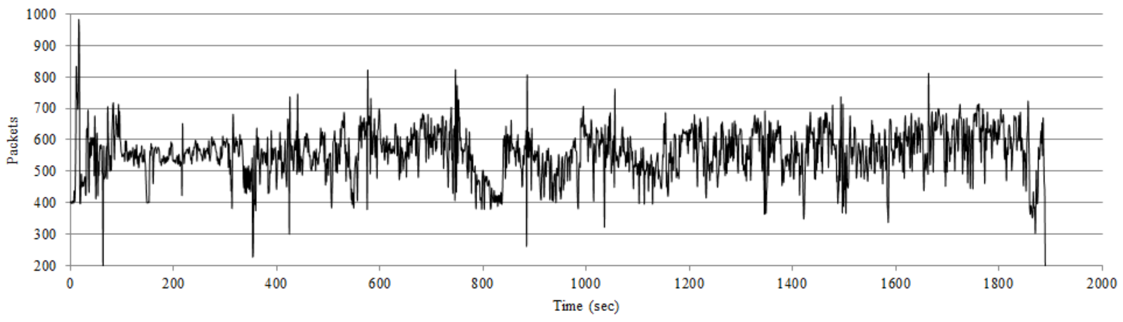
Figure 16: Session information extracted from datasets

ii) From the local interface, data was captured using Wireshark while playing video games on LAN and online using OnLive^{TM}. It was observed, after analyzing the datasets, that online servers can be caught up with demand and therefore employ load balancing. Load balancing refers to switching servers during the gameplay depending on load metrics. When a user starts to play a game, it may interact with a server which passes on the connection to another server. It was observed that server shifts happen randomly and that for accurate analysis it was imperative to include sessions generated by all servers in the particular communication. The result of accumulation of all connected sessions can be seen in Figure 17.

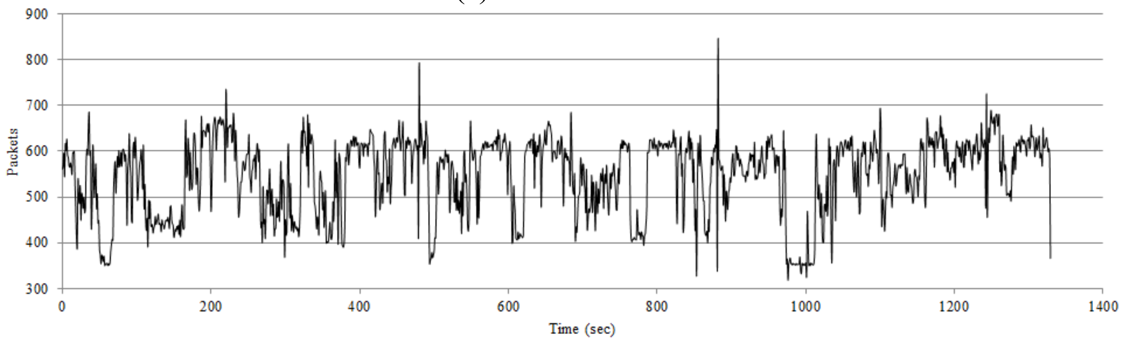
TABLE 2
Datasets/Traces

	Date	Size
ONLIVE		
Online	11-2011	846MB
“	11-2011	600MB
“	11-2011	559MB
Multiple Online datasets	10-2011	<500MB
LOCAL GAMES		
Multiple LAN datasets	09-2011	4-50MB
Counter Strike	2003	1-3MB
Command & Conquer	2003	1-3MB
Misc. Games	2003	1-3MB
CAIDA		
Chicago	10-2010	700MB
San Jose	03-2010	1.66GB
Chicago	02-2009	700MB
San Jose	03-2010	1.8GB
San Jose	12-2009	1.42GB
San Jose	03-2010	1.6GB
San Jose	12-2009	1.42GB
Chicago 100900	10-2003	1.07GB
Chicago 095900	11-2003	1.1GB
Chicago 100400	11-2003	1.06GB
NUST LOCAL		
Multiple LAN datasets	2008	8-10MB

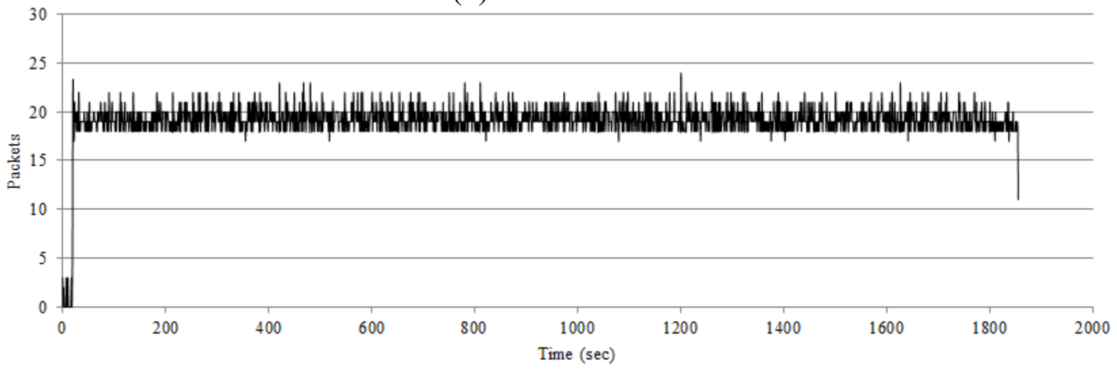
These datasets come under the category of SSR in Figure 17(a – b) and under the category of CSR in Figure 17(c – e). Observations and critical communication parameters such as packet length, arrival rate and average session length from these datasets are used in experiments for SBA in chapter 4.



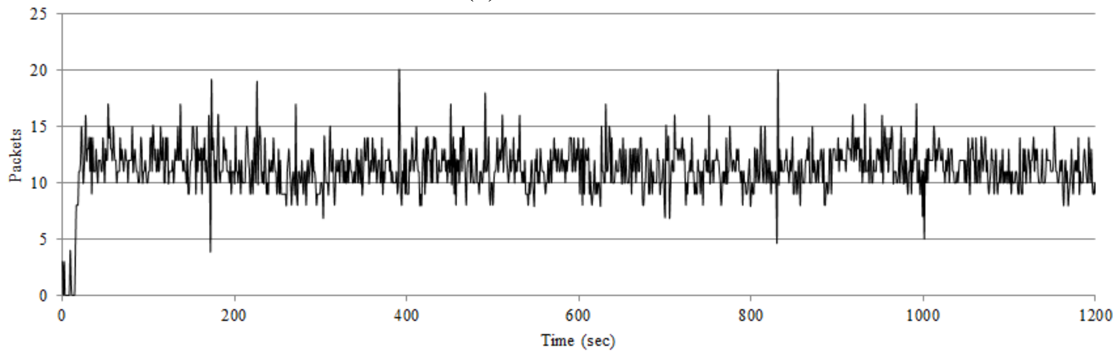
(a) OnLive trace



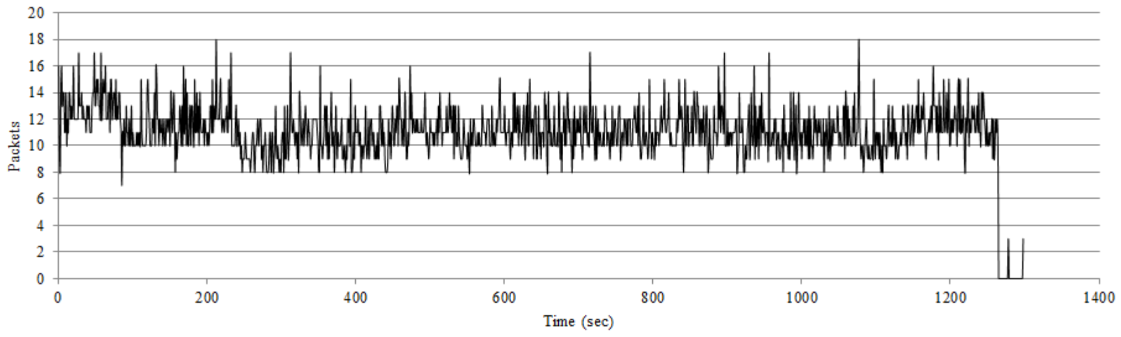
(b) OnLive trace



(c) LAN Trace



(d) LAN Trace



(e) LAN Trace

Figure 17: Video game datasets for SSR (a,b) and CSR (c – e) based games

Datasets shown in Figure 17 represent a sample of datasets that were collected. From Figure 17 once can see that data rates or arrival rates vary depending of the type of video-gaming experience i.e., SSR or CSR. For instance, as an example, for one of the OnLive datasets (SSR), average packet arrival rate of ~600 packets per second was observed with average packet size of ~800 bytes. This was low frame rate game play with sub HD resolution. The same game when played in Full HD and 50fps increases the packet arrival rate to ~1000 – 1500 packets per second. For CSR based games, bandwidth requirement is less than SSR since vectors do not require the same amount of space as frames. Packet arrival rates of ~100 packets per second with average packet size of ~200 bytes were observed for most datasets.

3.4 Discussions and remarks

This chapter discusses the data used for simulations that are detailed in chapter 4. Datasets or traces that were used for the purpose of analysis and simulations were gathered from a variety of sources. Large datasets containing information from a large number of users were downloaded from CAIDA [8]. These datasets give an insight into the behavior of internet traffic and general protocol stack. Service oriented, video-game related datasets were downloaded during live gameplay from online video gaming services such as OnLive [1]. LAN datasets were captured during gameplay on the local area network. Some older traces were also used to observe the difference in traffic generated by older and newer games. Captured and gathered datasets provide invaluable information regarding data generated by game engines in both SSR and CSR based systems as discussed in this chapter. It is evident from the graphs presented in this chapter that SSR generates much larger data compared to CSR. Also, communication sessions captured from high-speed interfaces in Chicago and San Jose show that data generated by individual users is in the form of bursts. Data sessions follow Pareto distribution which forms a basis for analytical modeling the next chapter.

CHAPTER 4 – BANDWIDTH ALLOCATION

In this chapter, latency analysis of ratio-counter based bandwidth allocation (RC-DBA) algorithm [46] for conventional Ethernet passive optical networks (EPON) [47] is carried out. Specific to the investigations, a novel bandwidth allocation algorithm, state-full bandwidth allocation (SBA) algorithm, for in-flight high-fan-out communication systems is also defined and analytical model is presented. Real-life video gaming data traffic is used for analysis and validation of design.

Since the advent of video-gaming industry, bandwidth requirements have been constantly increasing owing to ever-growing level of details in the game play and high-resolution graphics. Also, as the processing power increases, more processing-intensive gaming engines are being developed. At the heart of an IFE system is an efficient bandwidth allocation algorithm that is capable of providing high-speed content distribution with minimum to no starvation. Optical networks (ON), due to their light-weight, robustness and high-capacity, and being highly secure and non-intrusive medium, lower maintenance and future proof infrastructure, are very suitable for such an environment. Figure 1 in chapter 1 gives a generalized view of an in-flight video gaming system architecture that utilizes an on-board optical network for distribution of video game content, over an adapted schematic. Insets show placement of small footprint servers and optical splitters for distribution of contents to passengers.

For the purpose of analysis, data generated by video game servers has been divided into two categories, as shown in Figure 18 below, pertaining to the nature of video game server architectures. These categories are: A) Server side rendering (SSR) and B) Client side rendering (CSR).

A. Server side rendering

In a "server side rendering" (SSR) system, user interaction with the video game server happens at the server, as shown in Figure 18a. This means that data processing ensues at the server. OnLive [1] is an example of such a video gaming experience. With a global shift to cloud computing, it would not be wrong to predict that in the future most video gaming will be done in this way. There are some definite benefits of this approach. The most noteworthy advantage is that it works like video streaming which makes it useable on practically any computer that can stream video although, higher frame rates will require more processing power and bandwidth. All the processing is performed at the server end. Such a gaming environment produces multiple data streams often originating from multiple servers (as a result of load balancing) and therefore all new connections that are generated during the course of the game have been included in the analysis and design.

B. Client side rendering

"Client side rendering" (CSR) refers to a more conventional mode of playing a video game whereby client has the video game engine installed on the console and mainly motion and environment change-vectors are transmitted over the network, as shown in Figure 18b. Xbox^{TM} and PlayStation^{TM} are examples of client side rendering systems, although computers are also widely used as well. Server hosts the game and interacts with the consoles. Reasonably high processing power is required at the user end, and relatively small amount of data is transmitted over the network which makes this approach less bandwidth intensive but at the same time more expensive for IFE systems and less future proof.

Both categories are considered for the analysis in later sections. For demonstration a large bandwidth capacity provided by optical networks is used.

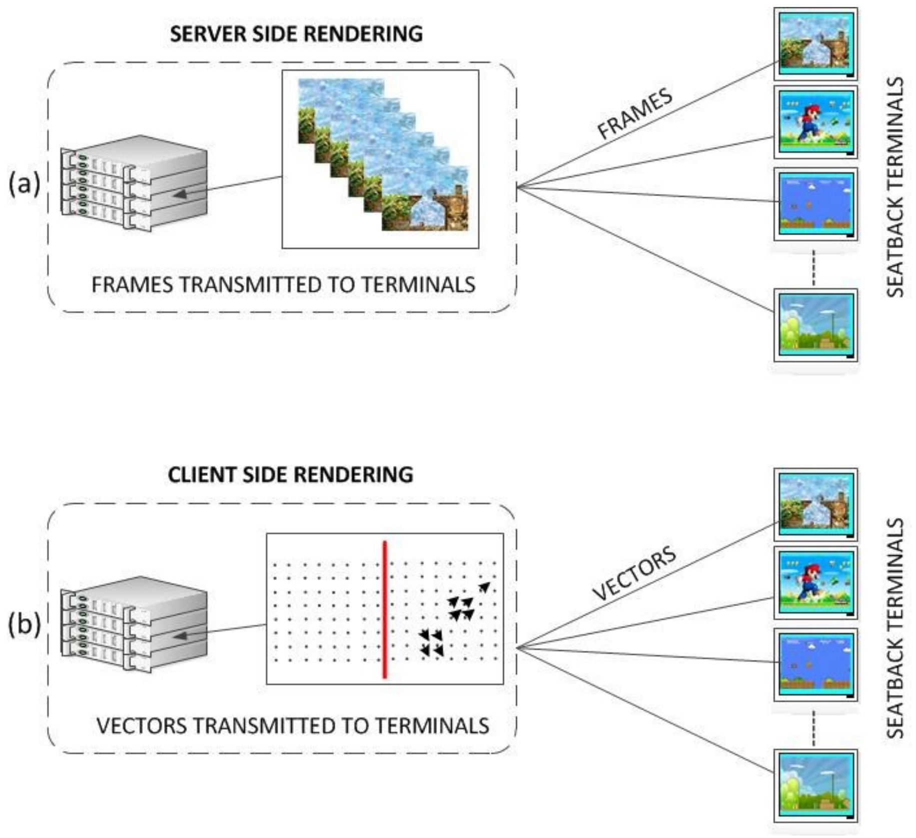


Figure 18: SSR vs CSR (a) Server side rendering: game play is processed at the server and frames are transmitted to the user (b) Client side rendering: game play is processed at the client end. Motion vectors and environment vectors are transmitted over the network.

4.1 Data collection and the datasets used

For the investigation and evaluation purpose, data is gathered from two sources; as a first step traffic behavior for both categories mentioned earlier, in this chapter, is observed. First source of datasets is OnLive [1] video gaming server which falls under the category of SSR. From the datasets resulting from server side rendering games, we observed regular video transmission data rates in the range of 22 – 29 Mbps and in some cases 33 Mbps for frame rates in the range of 45 to 60 fps in full HD or (1080p). This is considered high frame rate scenario. Second source was the datasets collected through network gaming and fall under the category of CSR i.e., client side rendering. Samples from these traces [1] are shown in Figure 19.

SBA was tested for a maximum of U=400+ users. According to the collected datasets (see 3.2 for details of datasets), on average, every user receives 550 – 560 packets/sec, with an average packet size of 805 – 814 bytes. This gives a maximum rate of 3.5Mbps. IP and UDP header lengths (base lengths) combine to be 32 bytes. Again, for the purpose of analysis, a quad-core processor with frequency of 3.0 GHz provides service rate of 12Gbps.

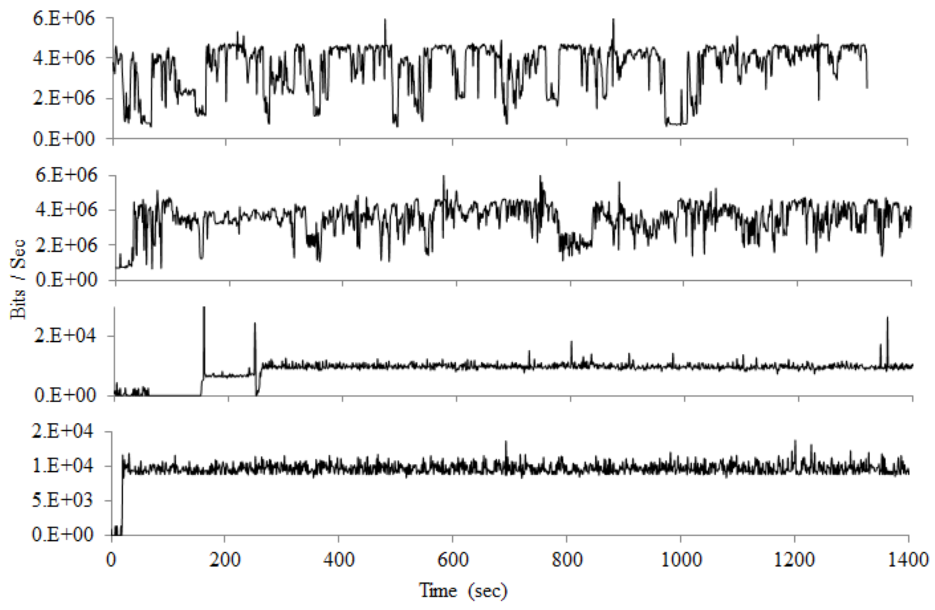


Figure 19: Video game reference traces (samples taken from larger datasets)

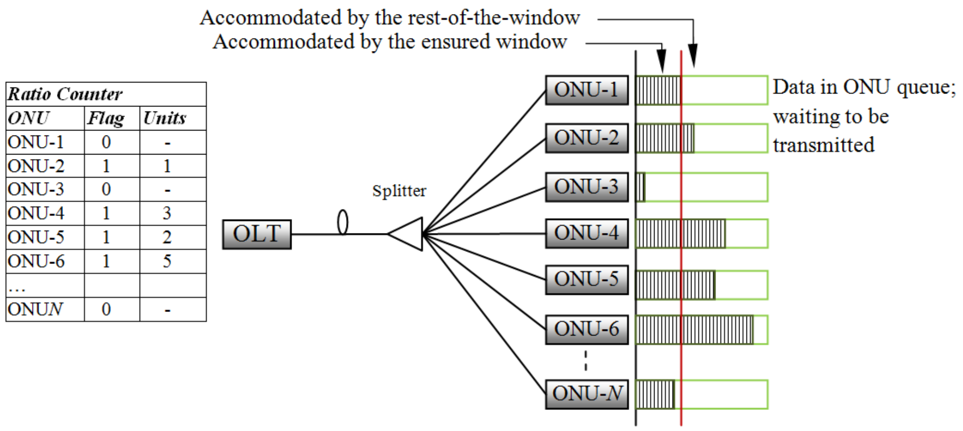
4.2 Bandwidth allocation for PON

Before discussing about bandwidth allocation for low-fan-out networks such as traditional PON, a short introduction about PON seems necessary. PONs or Ethernet PONs have shaped the way consumers access information over the internet by enabling high-speed data access at homes, offices and marketplace. A typical PON connects a server or a set of servers with the clients using a passive splitter as shown in Figure 20a. Optical Line Terminal (OLT) resides inside the server and Optical Network Unit (ONU) resides at the client end. Downstream is broadcast whereas in upstream, each ONU gets a time-slot to transmit on the shared channel. PONs use one dedicated wavelength for upstream communication and one other for downstream communication. Transmission from ONU to OLT is carefully timed so that each ONU gets a time-slot to transmit. ONUs communicate their requirements to the OLT and OLT grants their requests using pre-defined messages called primitives. Bandwidth allocation (BA) in a typical TDM fashion (i.e., fair) wastes considerable amount of bandwidth because if a contending ONU does not have any data to transmit, the allocated time-slots go wasted.

Various algorithms have been proposed to increase the bandwidth allocation efficiency in passive optical networks [46, 48-51]. Some focus has been shifted to bandwidth allocation in specialized networks such as long-reach PONs [50], allocation of resources in conventional PONs, 10Gbps PONs [48, 49, 51] and beyond, still remains a concern due to decreasing latency requirements. Access networks require larger bandwidth with users consuming huge bandwidth. This trend is increasing by the day. For networks that cater to a large number of active users, bandwidth allocation is even more critical and is discussed in section 4.3.

This section presents a novel analytical model to gauge performance of ratio-counter based dynamic bandwidth allocation (RC-DBA), in conjunction with the broader scope of progressing towards performance analysis of bandwidth allocation for high-fan-out networks described in the next section. RC-DBA is based on EFDBA [52] and improves on utilization and residual queue length [46].

RC-DBA, that was proposed earlier, algorithm allocates bandwidth (in time) to ONUs that wants to transmit in a particular time slot. During a communication session, each ONU communicates to the OLT, its buffer size. Buffer size represents the amount of time (translated from number of bytes sent in the request) required by the ONU in the next cycle. Average waiting time is the time between consecutive transmissions for an ONU. Cycle time is the time slot within which, all requesting ONUs get to transmit. Generally, there is a guard time between two cycles which prevents frames overlap in transmission sequence.


Accommodated by the rest-of-the-window
Accommodated by the ensured window
OLT
Splitter
ONU-1
ONU-2
ONU-3
ONU-4
ONU-5
ONU-6
ONU-*N*
Data in ONU queue; waiting to be transmitted

Ratio Counter		
ONU	*Flag*	*Units*
ONU-1	0	-
ONU-2	1	1
ONU-3	0	-
ONU-4	1	3
ONU-5	1	2
ONU-6	1	5
…		
ONU*N*	0	-

Figure 20(a): An overview of RC-DBA ratio counter and allocation table

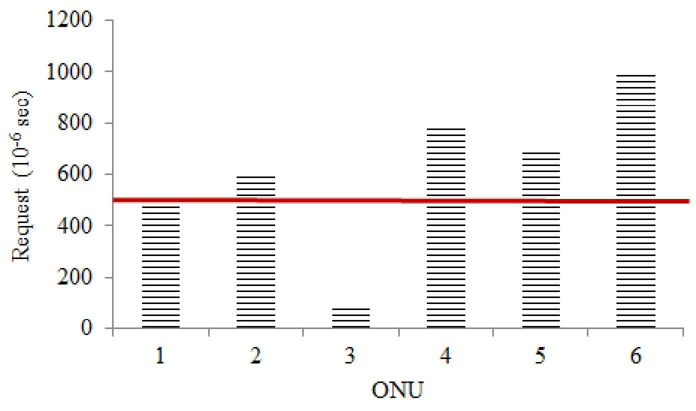


Fig 20(b): Requests from the connected ONUs. Red line represents the amount of bytes than can be accommodated within the ensured window

RC-DBA [46] divides a regular transmission cycle in two portions as shown in Figure 21. First portion is the "Ensured Window" (EW) that is allocated statistically based on the number of ONUs irrespective of whether it has data in its buffer or not. Later on, the time slot for this ONU (if idle) may be allocated to another ONU which needs more time. Second portion or rest-of-the-window (ROTW) is allocated solely based on requirements.

Suppose each ONU is allocated 500µs in the EW as shown in Figure 20b by the red line. In the given scenario ONU-1 has data (in buffer) that requires 200µs of transmission time slot. RC-DBA, allocates 500µs to the ONU-1 and moves to the next ONU. ONU-2 has data that requires 600µs to be transmitted, it is allocated 500 microseconds in the EW and over-load flag is set. Over-load flag identifies need for more transmission time. ONU-3 only has 100µs duration of data and is allocated 500 microseconds. This means that ONU-2 requires time in the ROTW. Similarly, all ONUs are allocated time slot in the EW. When EW is allocated completely, residual time is calculated. Residual time is the time-window that remains unused in the EW due to fewer requirements, such as in the case of ONU-1 and ONU-3. This way EW is squeezed and ROTW is broadened as illustrated in Figure 21 (bottom). ROTW can then be divided among ONUs that require more than what they were given in the EW.

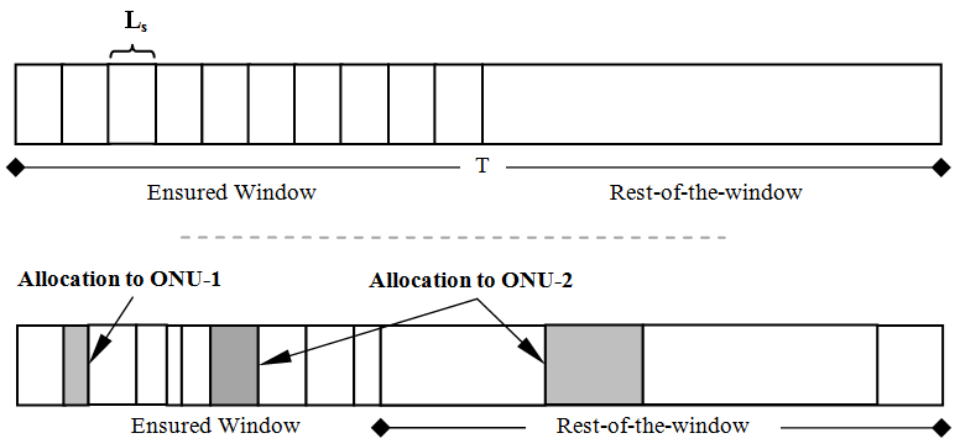


Figure 21: Cycle time distribution in RC-DBA. Pre-allocation state (top) and post-allocation state (bottom)

Allocation of ROTW works using a ratio based counter. A counter keeps track of how much time is required by each ONU e.g., ONU-2 requires 100µs extra slot. ONUs 4, 5, and 6 require 300, 200, and 500µs in ROTW, respectively. Ratio counter sets 100µs as its unit allocation for this cycle and then allocates multiples of 100µs. Therefore, ONU-2 gets one unit and other ONUs get 3, 2, and 5 units, respectively. Unit allocation may vary from cycle to cycle and is therefore adaptive. I has been shown earlier in separate work that this technique reduces residual queue length and ensures maximum channel utilization as shown in [46] and it was presented recently that the latency performance [7] of RC-DBA is better than other techniques. Next section discusses a more advanced way of allocation bandwidth.

4.3 State-full Bandwidth Allocation (SBA) for High-fan-out Optical Networks

Bandwidth allocation becomes far more involved when dealing with a large number of users. Along with added resource allocation challenges, there exists a dire need for enabling to offer services based on service level agreements (SLA). Some users may require more bandwidth than the others and this should be taken into consideration when allocating the resources. There is not much in the literature recently in terms of allocation in high-fan-out service oriented networks. Most of the related work is concentrated on passive optical networks that are discussed in the previous section.

The proposed SBA allocation is an advanced form of allocation that is based on evolving states within the system. State-full allocation is therefore a novel idea for allocation resources in optical networks.

TABLE 3
Weight Table - SBA

u	$u(\#,\omega)$	Hard-coded
1	$u(1,1)$	0
2	$u(2,3)$	1
3	$u(3,2)$	0
…		
U	$u(U,2)$	0

"State-full" refers to information gathered from previous states of the system where each cycle is considered to be a distinct state. Based on state information weights are assigned to users. However, weights can also be hard-coded for QoS assurance i.e., for the business class users/passengers or a high priority user sitting in economy class (in the context of IFE). Information from previous states is also used to predict of forecast upcoming requests from users. Since each user is assigned a weight, accordingly a weight or a priority code, accordingly a "weight table" (Table 3) is maintained. Assignment engine only takes information from weight tables in order to maintain data integrity in the system. Novelty in the technique lies in its state-full nature and ability to cater for high fan-out. Formally, as noted earlier the algorithm is referred to as State-full Bandwidth Allocation (SBA).

Numerous techniques have been proposed and reported [46, 48-50, 53, 54] over time for allocation of bandwidth in optical networks; however, high-fan-out state-full bandwidth allocation has not been tried and is novel. In the previous section we explained RC-DBA and our latency analysis of RC-DBA is reported in [7] for conventional passive optical networks i.e., low fan-out. As noted earlier, "high-fan-out" refers to a large number of users connected to the system considering a distribution system as a "black box". In later sections, analytical model for SBA is presented in order to incorporate bandwidths up to 100Gb/s from a single channel and support for high-fan-out. SBA focuses on single channel transmission systems, for now, that have been demonstrated to work at above mentioned data rates [55], however, with minimal changes it can be adopted to multi-channel systems as well.

4.4 Experiments

An analytical model was developed for SBA and latency analysis model was developed for both RC-DBA and SBA. In the following sub sections, parameters and configurations are defined and explained. Modeling is followed by residual time calculations and waiting time analysis in the later part of this chapter.

4.4.1 Analytical Model

This section describes analytical modeling of SBA. R represents incoming requests from users in consecutive cycles. $R_{\psi,u} = [\{R_{1,1}, R_{1,2}, R_{1,3}, \ldots, R_{1,N_U}\}, \{R_{2,1}, R_{2,2}, \ldots, R_{2,N_U}\}, \ldots]$, where ψ represents current allocation cycle, u is the current user and $R_{\psi.u}$ is the request by a user in current cycle where, $u \in |\mathbb{Z}||$ and $\psi \in |\mathbb{Z}|$. All requests are in the time domain. $C \in |\mathbb{Z}|$ is the total number of cycles in the communication session. Total requests in current cycle are given by Eqn. (1) and the total requested time for an entire communication session is given by $R_C = \sum_{i=0}^{C} \sum_{j=0}^{U} R_{i,j}$. $\gamma_{FA,u}$ is the maximum amount of time that may be allocated in first allocation to a user u i.e., allocation threshold. This allocation increases based on need for more requirement for a set of particular users in the secondary allocation. There are three possible scenarios: i) $R_{\psi,u} \leq \gamma_{FA,u}$ ii) $R_{\psi,u} = \gamma_{FA,u}$and iii) $R_{\psi,u} > \gamma_{FA,u}$. Basis of this allocation is the request from user informing the server about the amount of time it requires in the next cycle. Depending on the weight assigned to each user at the time of network initializing, a user will have more time allocated to it in $\gamma_{FA,u}$, however, this allocation is within the bounds $0 < \varphi_{FA.u} \leq \gamma_{FA,u}$ where $\varphi_{FA,u}$ is the time allocated to user u in first allocation. The maximum cycle time T_C must not be exceeded therefore the limit holds,

$$R_{\psi}(\psi_{FA}) = \sum_{i=0}^{U} R_{\psi,i} \leq T_C\, , R_{i,j}, 0 < \varphi_{FA,u} \leq \gamma_{FA,u} \tag{1}$$

where $\psi_{FA,u}$represents first allocation to user u in the current cycle. Also, to avoid starvation we ensure minimum bandwidth (time-slots) to every user. This is represented by $\gamma_{MIN,u}$ and every user is entitled to this much allocation in every cycle. Table 4 lists all parameters (with definition and notation) used in this paper for a quick reference.

A. Initialization of $\gamma_{MIN,u}$, T_C and Cycle set (N)

Minimum allocation, $\gamma_{MIN,u}$, is determined based on the number of active users and weights set for each user. It is calculated during initialization and stays the same thereafter. Figure 22a shows a snapshot of a cycle. A user with low weight gets a smaller minimum time window compared to a high weight user (see Figure 22b).

Similarly, a high priority/weight user gets to transmit for a larger amount of time in the current cycle (Figure 22c) compared to a low weight user (Figure 22e). All users get the same weight in the first cycle and system stabilizes after initial cycles. In a real world scenario, for instance on Airbus A-380 aircraft, 8 business class passenger may be assigned weight “1”, 80 first class passengers may be assigned weight “2” and rest of the

450 passengers may be assigned weight “3”. Assigning the same weight to each user initially tests the bandwidth allocation technique with maximum dynamic allocations.

TABLE 4
Parameters

Symbol	Quantity
$R_{\psi,u}$	Request by a user in current cycle (sec)
ψ	Current transmission cycle
ψ^*	Previous transmission cycle
u	Current user
U	Total number of users
C	Number of transmission cycles in a communication session
R_ψ	Total request for current cycle (all users)
R_C	Total request for entire communication
$\gamma_{FA,u}$	Maximum allocation in 1st iteration for user u
$\gamma_{MIN,\omega}$	Minimum allocation for weight group w in a cycle
$\gamma_{SA,u}$	Maximum allocation in 2nd iteration for user u
$\varphi_{FA,u}$	Time allocated to a user in first allocation
$\varphi_{SA,u}$	Time allocated to a user in second allocation
T_C	Maximum cycle time
T_G	Guard time
$u(\#,\omega)$	User (#) with weight w
ω_u	Weight assigned to user u
r	Data rate (100Gb/s)
N	Number of cycles in a cycle set
τ_g	Unit-segment of time / unit of time for allocation in first and secondary allocation

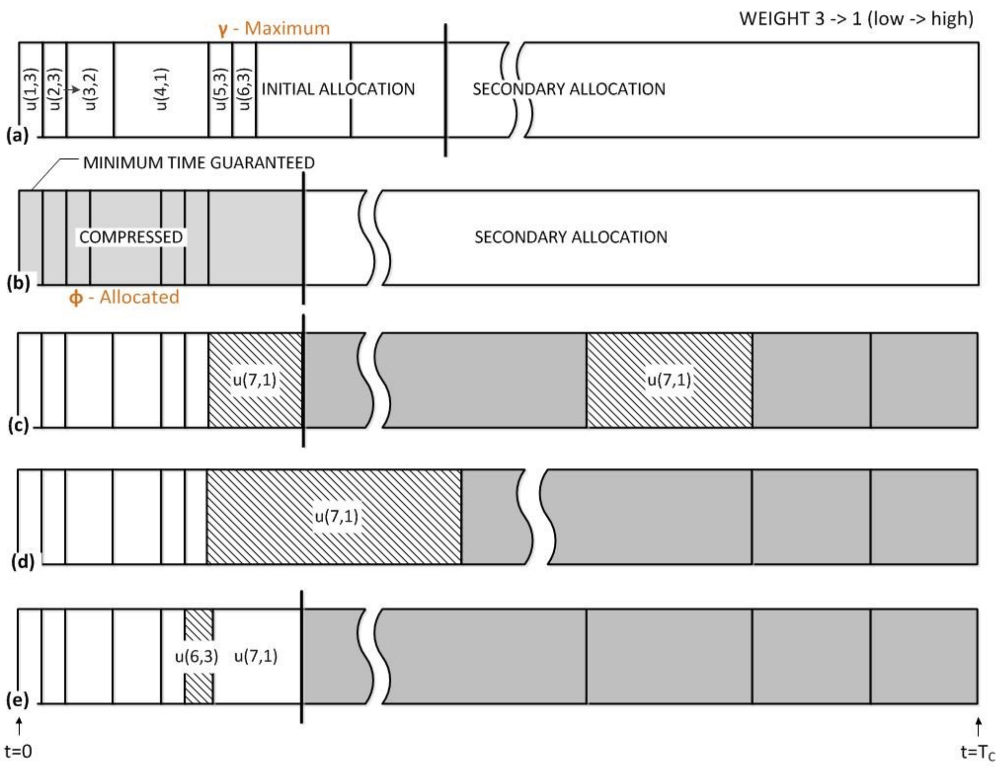


Figure 22: Bandwidth allocation in SBA (a) Initial allocation based on weights (b) Secondary allocation (c) Secondary allocation for user 7 (d) Efficient single combined slot for user 7 (e) Secondary allocation for user 6. Initial allocation (left of bold vertical line) and secondary allocation (right of bold vertical line)

Cycle time T_C is fixed for a cycle set. Optimal cycle time depends on data rate and the number of active users using the network. Latency increases with growing cycle time [7]. A “cycle set” as shown in Figure 23 is the group of cycles for which systems allocation parameters (i.e., weights) remain unchanged. Parameters such as weights are re-assigned after a cycle set is over; therefore it is called a dynamic cycle set.

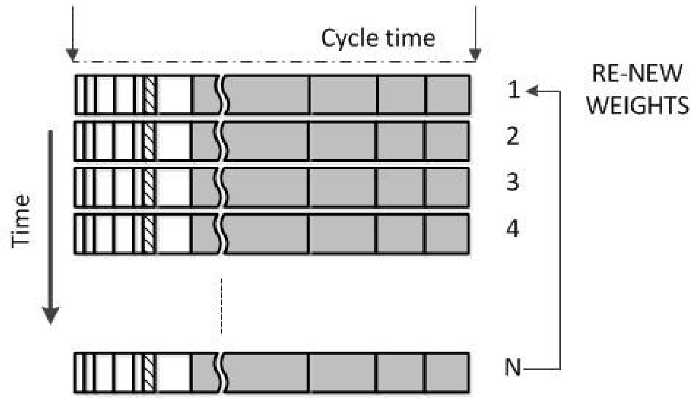


Figure 23: A dynamic cycle set

B. Initialization of first allocation ($\gamma_{FA,u}$)

In a fair system of allocation, each user carries the same weight. Fair systems of allocation are not very efficient when it comes to large number of users due to wastage of bandwidth and increased latency. Amount of wasted bandwidth can be reduced by assigning weights to users individually. A user with a higher weight gets a larger chunk of bandwidth. System assigns weights to users based on their past requests over the last cycle set. For assignment, we consider the following cases:

i) $R_{\psi^*,u} \leq \gamma_{FA,u}, \psi^* \in [(\psi-1),(\psi-2),\ldots,(\psi-n)]$: User has been requesting less than the threshold in the last N cycles. A critical parameter N, which represents the number of previous states we consider while making the assignment decisions for the current cycle ψ.

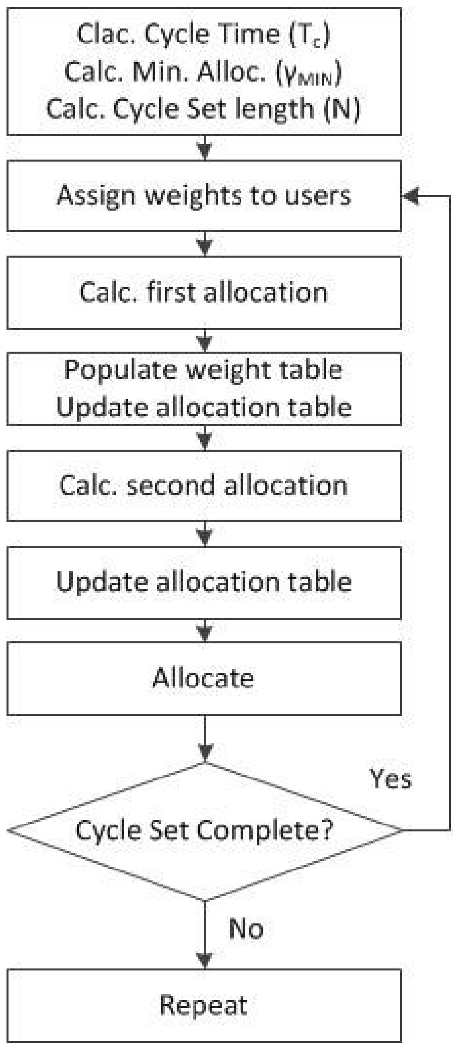


Figure 24(a): Flow chart for allocation in SBA

ii) $R_{\psi^*,u} > \gamma_{FA,u}, \psi^* \in [(\psi-1),(\psi-2),\dots,(\psi-n)]$: User has been requesting more than the threshold in the last N cycles.

iii) $R_{\psi^*,u} = 0, \psi^* \in [(\psi-1),(\psi-2),\dots,(\psi-n)]$: User has not requested anything in the last N cycles, at all.

Since, decision for each user is taken individually; each user can be assigned a weight $\omega \in P$, where $P = [1,2,3]$. From the next section onwards we will use $u(\#,\omega)$ to represent a user with a weight ω. $\gamma_{FA,u} \geq \gamma_{MIN,u}$ must be maintained at all times to avoid starvation. Low weight users get to transmit in

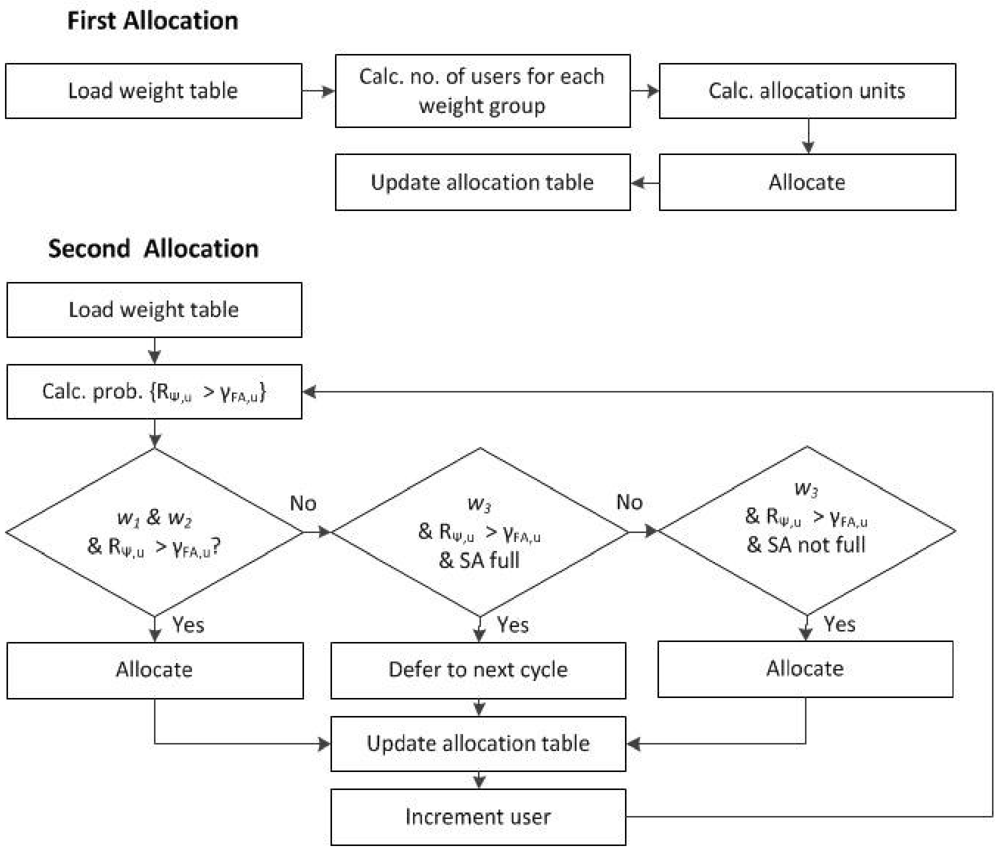

First Allocation
Load weight table
Calc. no. of users for each weight group
Calc. allocation units
Allocate
Update allocation table
Second Allocation
Load weight table
Calc. prob. {$R_{\psi,u} > \gamma_{FA,u}$}
w_1 & w_2 & $R_{\psi,u} > \gamma_{FA,u}$?
No
w_3 & $R_{\psi,u} > \gamma_{FA,u}$ & SA full
No
w_3 & $R_{\psi,u} > \gamma_{FA,u}$ & SA not full
Yes
Yes
Yes
Allocate
Defer to next cycle
Allocate
Update allocation table
Increment user

Figure 24(b): Flow chart: first and second allocation

Since, decision for each user is taken individually; each user can be assigned a weight $\omega \in P$, where $P = [1,2,3]$. From the next section onwards we will use $u(\#, \omega)$ to represent a user with a weight ω. $\gamma_{FA,u} \geq \gamma_{MIN,u}$ must be maintained at all times to avoid starvation. Low weight users get to transmit in the initial cycle if they have data to transmit. Weight can be assigned to a user based on the following two cases: i) hard coded ii) dynamic. User 1 with an assigned weight of 1 is represented as $u(1,1)$, as shown in Table 3, with a bit set to 1 if it is hard-coded. Hard-coded weights are not set or re-set dynamically. Weights can be dynamically calculated for each user as:

$$\omega_u = \frac{1}{N}\sum_{\psi^*=1}^{N}(R_{-\psi^*,u} \geq \gamma_{FA,u})$$

(2)

ω_u is calculated for all users in accordance with weight assignment case (ii) above. Hard-coded weights are set before dynamic weights are calculated. For first cycle all users, except hard-coded, are assigned equal weight. Therefore,

$$x(\omega_u) = \begin{cases} 1, & 0.0 < \omega_u < 0.29 \\ 2, & 0.3 < \omega_u < 0.69 \\ 3, & 0.7 < \omega_u < 1 \end{cases}$$

(3)

Weight table is shown above in Table 3. If a user requests more than threshold constantly in previous N cycles, it is allocated a higher priority for next N cycles (a cycle set, Figure 23). In this manner we can ensure that bandwidth is allocated according to requirements and also that it is not wasted. Flow chart of the algorithm is shown in Figure 24a and 24b. Cycle sets with dynamically changing weights are shown in Figure 25, where dark bars represent changes within the consecutive cycles. These changes reflect the dynamic nature of weight assignment.

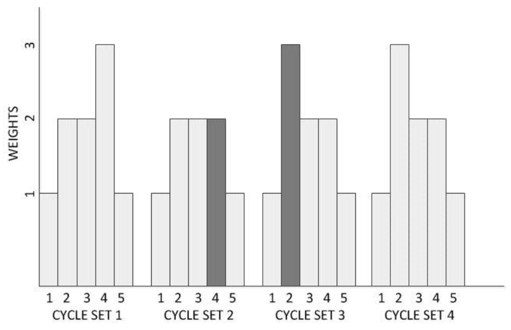


Figure 25: Changing cycle set

We use Pareto distribution [56] that represents the distribution of requests from users in a cycle. Pareto distribution is characterized by a sharp initial rise for small duration and falling a long tail. It follows from the famous 80/20 rule of economics [57] and fits really well for internet traffic models. On the basis of analysis we determine, $\Pr(R_{\psi+1,u} > R_{\psi-1,u})$, which is the probability that a user will request greater than its request in previous cycle. If it's high, this probability means that user will be a contender in secondary allocation. Considering the number of requests received from users as Pareto distributed we can define the probability that a user will have request in the next cycle $(\psi + 1)$, as given by Eqn. (4).

$$\Pr(R_{\psi+1,u} > R_{\psi^*,u}) = \begin{cases} \left(\dfrac{x_m}{R_{\psi^*,u}}\right)^{\alpha}, & R_{\psi^*,u} \geq x_m \\ 1, & R_{\psi^*,u} < x_m \end{cases}$$

$$\Pr(E(X)_u) = \frac{1}{N}\sum_{i=1}^{N} \Pr(R_{\psi+i,u} > R_{\psi-i,u}) \tag{4}$$

Since $\gamma_{FA,u}$ is maximum allocation allowed in first allocation for user u, initialization can be formulated based on parameters given in (i) – (iii) above. $\gamma_{FA,u} = \omega_u * \tau_{g,FA}$, where $\tau_{g,FA}$ is a unit-segment of the time that constitutes secondary allocation cycle. In the second allocation $\gamma_{SA,u}$, each user gets a time slot based on its weight.

C. Allocating time in secondary allocation $(\gamma_{SA,u})$

Initial cycle is compressed before allocating secondary cycle, such that $\gamma_{SA} = T_C - \varphi_{FA}$ where $\varphi_{FA} = \sum_{i=0}^{U} \varphi_{FA,i}$ and $\gamma_{SA} = \sum_{i=0}^{U} \gamma_{SA,i}$, therefore, $\tau_{g,SA} = \frac{\gamma_{SA}}{[\omega_1 * N_1]+[\omega_2 * N_2]+[\omega_3 * N_3]}$. If a user with high weight requires more time to transmit, it gets more time in secondary allocation whereas if a user with low weight requires more time, it has to wait for the next cycle if secondary allocation is complete. If secondary allocation is not complete and there is time to spare, a user with low priority will be accommodated in the interest of maximum utilization, as shown in Figure 24b. Also, if a user has not requested any bandwidth or has been requesting low bandwidth in the last N cycles, its weight reduces or remains unchanged. Slots can be distributed among contending users depending on weights and $\tau_{g,SA}$ as:

$$\varphi_{SA,u} = \omega_u * \tau_{g,SA} \tag{5}$$

D. Allocation Table / Assignment (SBA)

Allocation table keeps track of incoming requests, weights, and maximum allowed time, allocated time in first allocation, granted time in second allocation and residual time. For randomly generated requests arriving from users, assignment engine allocates the slots based on Figure 24a and 24b. Figure 26 shows a glimpse of simulated residual times (i.e., time taken forward to the next cycle due to lack of available time in current system) for SBA.

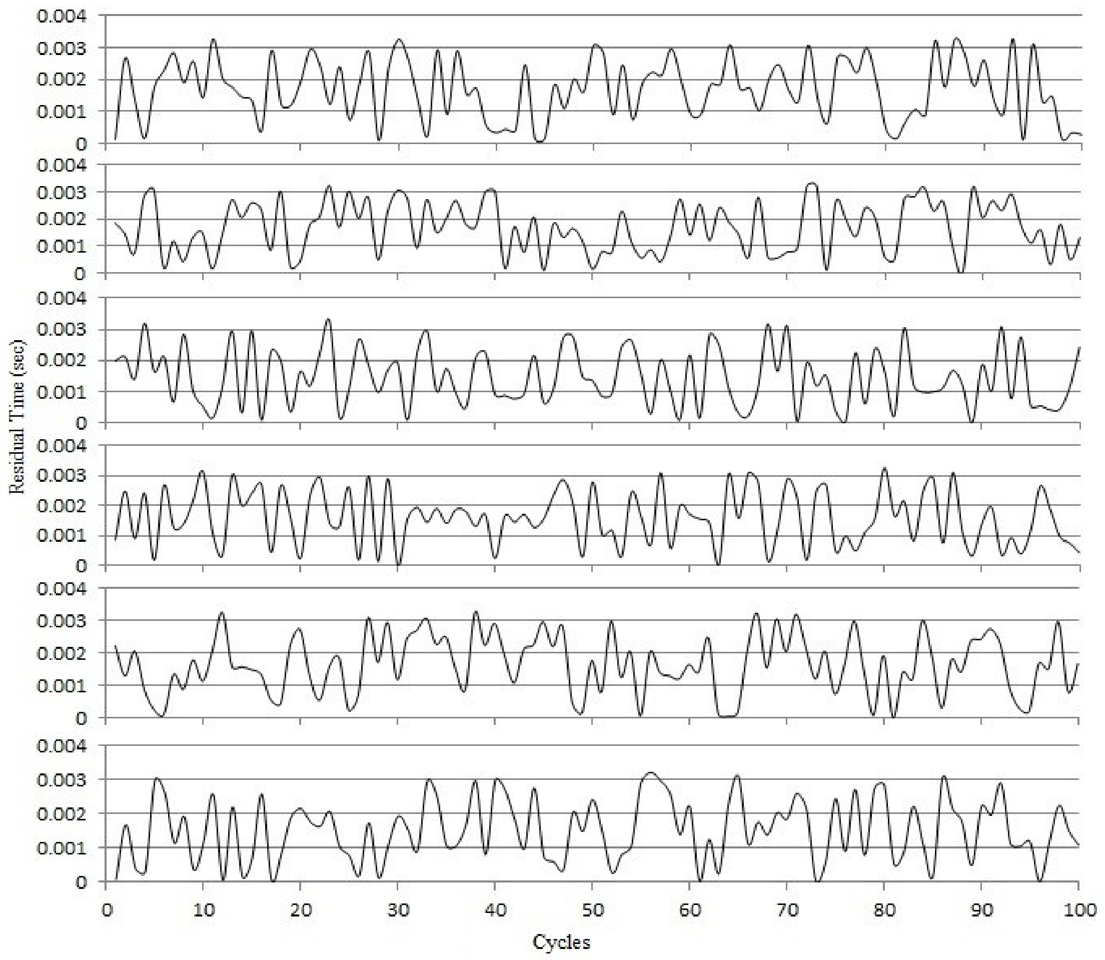


Figure 26(a): Residual time after every cycle for 100 cycles for random users

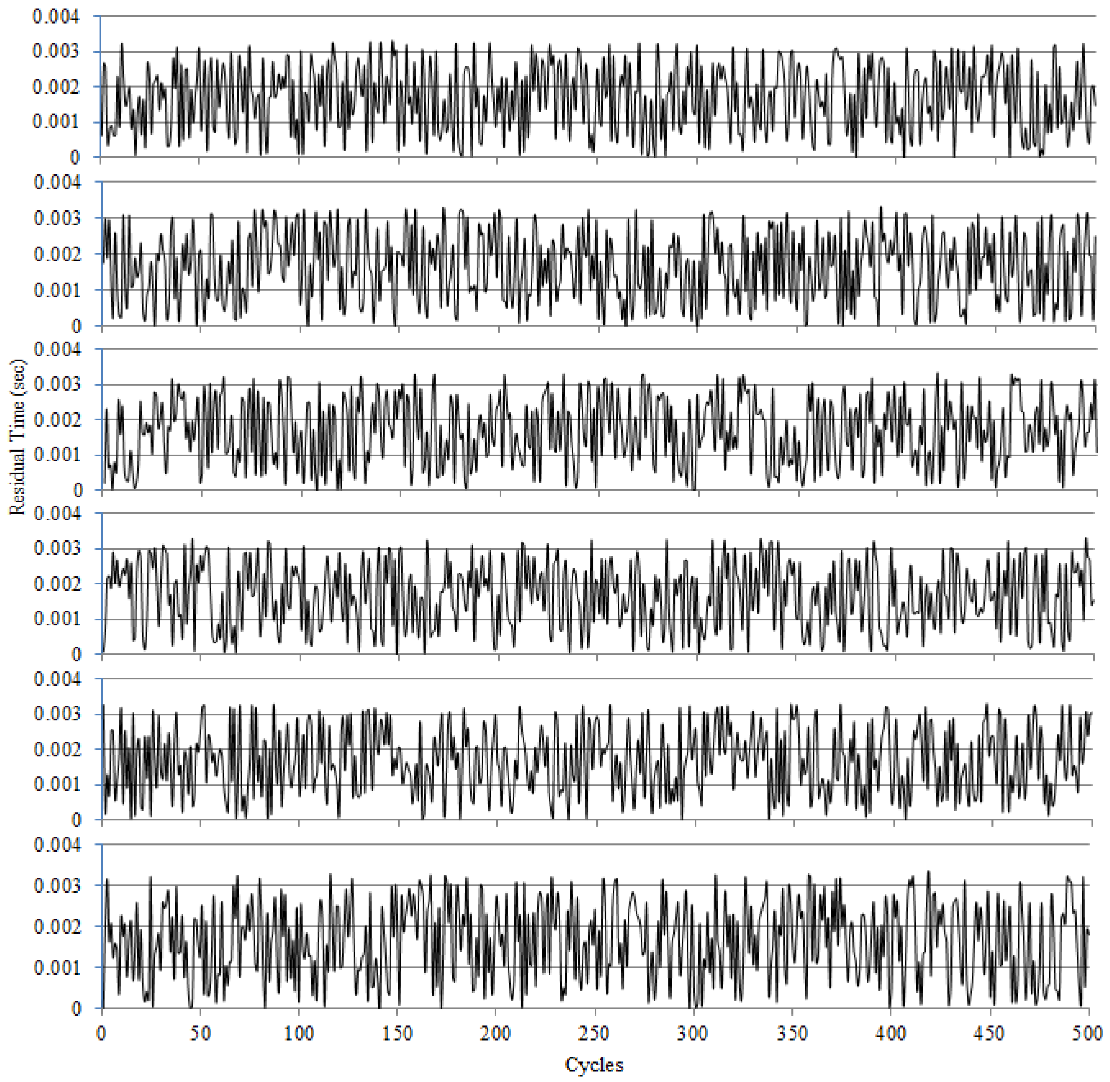


Figure 26(b): Residual time after every cycle for 500 cycles for random users

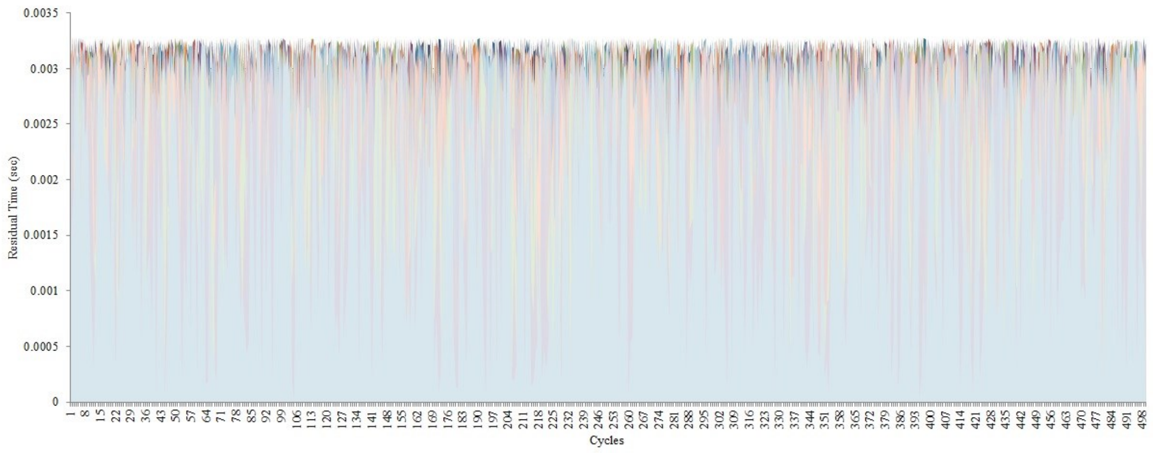


Figure 26(c): Residual time after every cycle for 500 cycles for 250 users

One can see that residual times remain in an acceptable range. Above shown results are for randomly selected users out of a total of 500 since pictorial representation of all users renders the plots unreadable as shown in Figure 26c. The results in Figure 26 are from simulations that ran for 100 cycles (a) and 500 (b) accumulating residual times along the way. We can see that residual time remained within the range. This is also the time during which system stabilizes. For a particular cycle, if a user requires less time (for new data) residual requests are serviced and therefore residual time at the end of the cycle decreases. This behavior is evident in Figure 26 (i.e., valleys).

4.4.2 Latency Analysis

This section presents latency analysis model of RC-DBA followed by that of SBA. Average waiting time for an ONU, between transmissions is calculated, for RC-DBA and compare it with EFDBA and statistical multiple access. For completeness, we compare Poisson as well as Pareto distributed arrivals. Equation (7) gives the delay incurred when request is fulfilled by EW. In this case, ROTW stays idle and bandwidth is wasted but since there is no data to be transmitted it cannot be allocated. We can however reduce the cycle time by half in case of low load and reduce the average waiting time by half. W_{HIGH} is the average waiting time experienced by an ONU under high load.

For the first scenario, average residual time before an ONU gets to transmit again can be expressed by Eqn. (6). T is cycle time which is kept at 2 msec. Theoretically, T can be any length i.e., T= [1ms, 2ms, 3ms,…], but length of cycle time affects waiting time in consecutive cycles for ONUs. Ensured and ROTW, initially, are equal to half the cycle time i.e., T/2 each. N is number of ONUs connected to the OLT and n is the number of computers or devices connected to the ONU such that N=[1,2,3,….N] and n=[1,2,3,….n]. EW is compressed, but not expanded, after the initial fair allocation. Compression in EW results from smaller requirements from ONUs. This spare time is allocated to highly loaded ONUs in the ROTW. Since average waiting time is the same for each ONU in the first scenario, i and j (current ONU and device number respectively) are irrelevant. T_G is the guard time kept at 2µs. Standard EPON data rate of 1.25Gbps has been used for all calculations.

$$R_{Low} = \begin{bmatrix} \left[\left\{(N-i)\left(\frac{T}{2N}\right)\right\} + \left\{(n-j)\left(\frac{T}{2N} * \frac{1}{n}\right)\right\} + \left(\frac{T}{2}\right)\right] + \\ \left[\left\{(i-1)\left(\frac{T}{2N}\right)\right\} + \left\{(j-1)\left(\frac{T}{2N} * \frac{1}{n}\right)\right\}\right] + T_G \end{bmatrix} \tag{6}$$

Waiting time can be calculated from [58] for the first scenario as shown in Eqn. (7) below.

$$\bar{W}_{LOW} = \frac{\left[\left\{T\left(1 - \frac{1}{2Nn}\right)\right\} + T_G\right]}{2\left(1 - \left(\frac{n\alpha x_m NT}{\alpha - 1}\right)\right)} \tag{7}$$

Where α is a positive parameter larger than 1 and x_m is the minimum value of arrival. We vary arrival rate to see the effect on waiting time. Mean values of Poisson and Pareto distribution have been used for the purpose of this analysis. Pareto distribution maps network (predominantly TCP) traffic pretty accurately. It is a long tailed distribution with small number of occurrences of large magnitude and a large number of occurrences of smaller magnitude. This behavior is consistent with network traffic and can be observed easily [24].

In the second scenario, ONUs have more data and therefore require ROTW. High load situation requires a more complex analysis since any ONU can request a slot in ROTW, randomly. This essentially means that there is a high probability that an ONU will have more data to transmit than allocated in EW. Average waiting time for such a scenario is given by Eqn. (8). Increase in incoming traffic causes the ROTW to be utilized efficiently, unlike in the low load situation where it goes largely wasted. The result of this is reduction in residual queue lengths. Average waiting time also reduces significantly since an ONU can have allocation in both ensured and ROTW of the same cycle depending on its initial request. A maximum of $\mathrm{T}/2\mathrm{N}$ may be allocated to an ONU in the EW.

$$\bar{W}_{HIGH} = \frac{\left[\left\{T\left(\frac{3}{4} - \frac{1}{nN}\right)\right\} + T_G\right]}{\left(1 - \left(\frac{n\alpha x_m NT}{\alpha - 1}\right)\right)} \tag{8}$$

Compared with EFDBA, RC-DBA shows smaller waiting times. Major reason for this behavior is the fact that ROTW is not allocated fairly, instead, only the ONUs that require data to be transmitted are allocated time and those that do not require even the EW slot are not given the whole slot. Eqn. (9) gives the average waiting time for EFDBA. This has been determined for the purpose of comparison.

$$\bar{W}_{EFDBA} = \frac{\left[T\left\{\frac{(2n(3N-2)-1)}{4Nn}\right\} + T_G\right]}{2\left(1 - \left(\frac{n\alpha x_m NT}{\alpha - 1}\right)\right)} \tag{9}$$

Ratio counter is a table (as shown in Table 5) enlisting the status of ONUs in terms to their requirement for a time slot in ROTW. Towards the end of for allocation cycle, the algorithm calculates all the remaining requests that still need to be fulfilled. The smallest such request is called a unit. Multiples of this unit is then allocated in ROTW. As shown in Table 3, ONU 1 needs a single unit whereas ONU 2 needs three units. ONU 3 does not need any further allocation in this cycle.

TABLE 5

RC-DBA

ONU	Request	Flag	Remaining	Units	Residual
1	500	0	0	-	0
2	600	1	100	1	0
3	100	0	0	-	0
4	800	1	300	3	0
5	700	1	200	2	0
6	1000	1	500	5	0
…					
N	450	0	0	-	0

An ONU that does not require a time slot in ROTW is not flagged. Only Flagged ONUs get unit allocation in ROTW. Any data that remains to be transmitted after each cycle is passed on to the next cycle for scheduling. Table 5 shows no residual data from this cycle to the next. 100 microseconds is the minimum request (remaining) for this cycle.

We can treat the ratio-counter as an array of random variables where each item in the array represents an ONU. These requirements are fulfilled as multiples of unit allocation that is calculated in every cycle and may change in every cycle. Also, there may be more units allocated in ROTW than the time remaining therefore all that can be transmitted is transmitted in the current cycle and residual requirement (if any) from all ONUs is forwarded to the next cycle and is added to the new requirement in that cycle.

For SBA however the system is modeled in a slightly different manner since weights and statistically calculated parameters are involved. Residual times are calculated and based on those, average waiting times are determined for users. Since average waiting time is the same for all users individual times need not be calculated for close approximation.

$\mathrm{N}_{\omega} = [\mathrm{N}_1, \mathrm{N}_2, \mathrm{N}_3]$ is a set containing the number of users with respective weights. $\mathrm{t}_\mathrm{a} = \sum_1^{i-1} \gamma_{\mathrm{MIN,u}}$, $\mathrm{t}_\mathrm{b} = \sum_{i+1}^{\mathrm{U}} \gamma_{\mathrm{MIN,u}}$ as shown in Figure 27. Slot length is equal to packet length and we neglect propagation delay due to negligible distances.

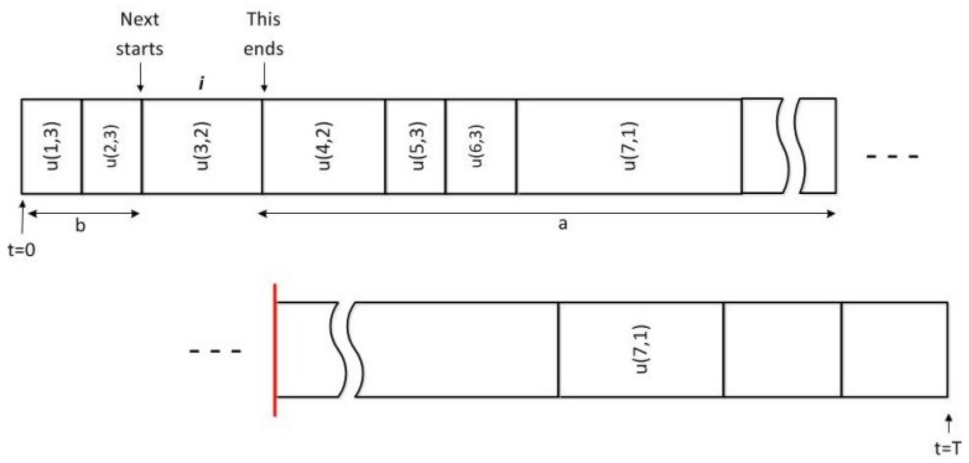


Figure 27: Residual time calculation (i.e., for user number 3)

$$E(X)_u = \begin{cases} 0, & \Pr(E(X)_u) < 0.5 \\ E(X), & \Pr(E(X)_u) \geq 0.5 \end{cases}$$

(10)

Residual time can be calculated as $R = \varphi_{FA} + \varphi_{SA} + T_G = t_b + t_a + \varphi_{SA} + T_G$ and residual time for user u when it does not qualify for a slot in secondary allocation is then,

$$R_u = \sum_{j=1}^{u-1} \varphi_{FA,j} + \sum_{j=u+1}^{U} \varphi_{FA,j} + \varphi_{SA} + T_G$$

(11)

Similarly, when user u gets a slot to transmit in secondary allocation based on Eqn. (10), it faces a different wait time until the next cycle since transmission time in the current cycle increases. Also, if a user gets time in secondary allocation, it is allocated the total time together in one slot in order to reduce management complications as shown in Figure 22d. In this case residual time is given by,

$$R_u = \sum_{j=1}^{u-1} \varphi_{FA,j} + \sum_{j=u+1}^{U} \varphi_{FA,j} + \varphi_{SA} - \varphi_{SA,u} + T_G$$

(12)

where, u is the current user and total allocation to a user in next cycle is $\varphi_{FA,u} + \varphi_{SA,u}$.

From Eqn. (11), for waiting time, we get,

$$W = \frac{\sum_{j=1}^{i-1} \varphi_{FA,j} + \sum_{i+1}^{U} \varphi_{FA,j} + \varphi_{SA} + T_G}{2\left[\frac{\alpha(1 - x_m UT) - 1}{\alpha - 1}\right]} \tag{13}$$

and from Eqn. (12), for waiting time, we get,

$$W = \frac{\sum_{j=1}^{u-1} \varphi_{FA,j} + \sum_{j=u+1}^{U} \varphi_{FA,j} + \varphi_{SA} - \varphi_{SA,u} + T_G}{2\left[\frac{\alpha(1 - x_m UT) - 1}{\alpha - 1}\right]} \tag{14}$$

Equations (13) and (14) are special cases of the average latency incurred by a user during and between cycles and can be derived from Figure 27.

4.4.3 Results

As mentioned earlier, cycle time of 2 ms has been set for RC-DBA and cycle time changes for SBA. Cycle time must not be too large or too small. For a large cycle time, average system waiting time increases and for small cycle times much of the time is spent in managing cycles compared to transmitting data which increases overhead. Also, if transmission window allotted to a particular ONU is less than the time it needs to transmit a packet completely, the packet will need fragmentation that would require processing as well as overhead transmission of fragmented headers. Figure 28 shows that waiting time increases exponentially as the cycle time increases beyond a threshold. This behavior is relatively consistent when we use larger number of ONUs. Figure 29 shows increase in delay as number of ONUs connected to OLT increases. This is consistent with the characteristic waiting time curve for TDMA systems. This is obvious since each

ONU is pushed back in time for next transmission due to time allocated to an added ONU.

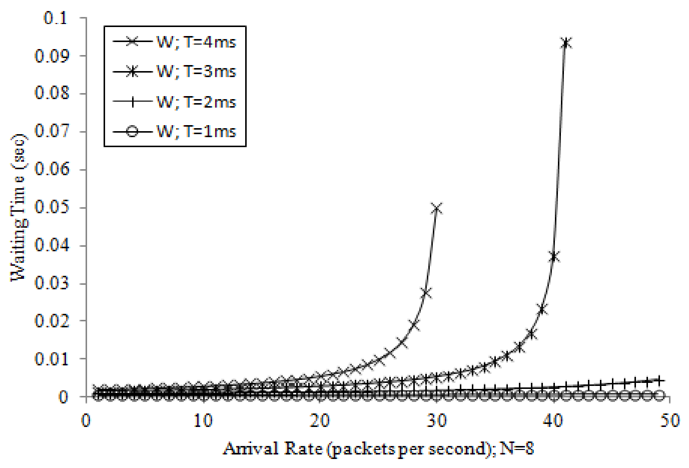


Figure 28: Waiting time increases with larger cycle time (T)

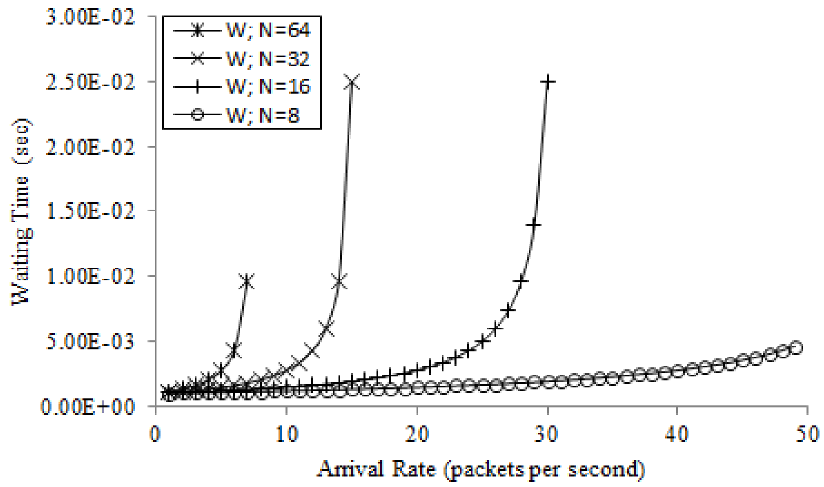


Figure 29: Waiting time increases with increase in arrival rate at ONUs

Compared with existing low-fan-out BA algorithms, RC-DBA performs better in many aspects as shown in figures below. Latency incurred by using RC-DBA is much less than that of statistical and much better than EFDBA. Figure 30a gives a comparison between EFDBA and RC-DBA. It can be seen clearly that RC-DBA out performs EFDBA in latency performance at least by three orders of magnitude. This is significant improvement and delay being reduced to microseconds means a packet can be transmitted in every transmission cycle although very small.

For completion and reference, statistical allocation incurs huge delays, as shown in Figure 30b, when the number of connected ONUs is large. Since allocation is fair there is not much that can be done to improve.

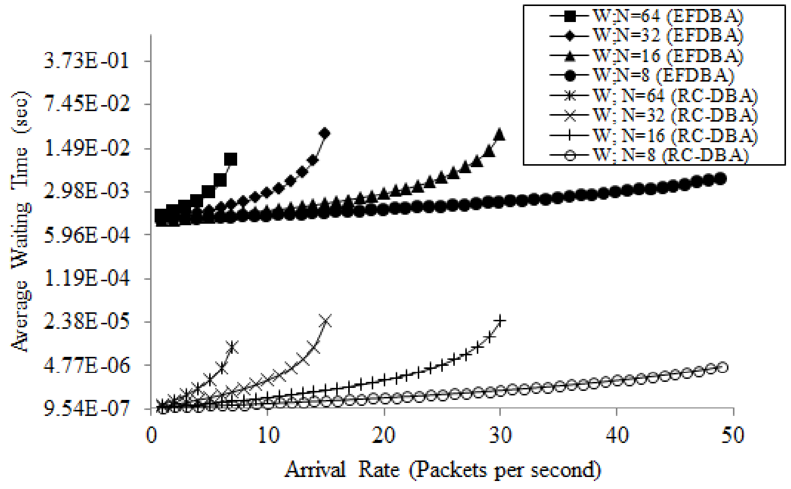


Figure 30(a): Waiting time vs Arrival rate for Poisson arrivals

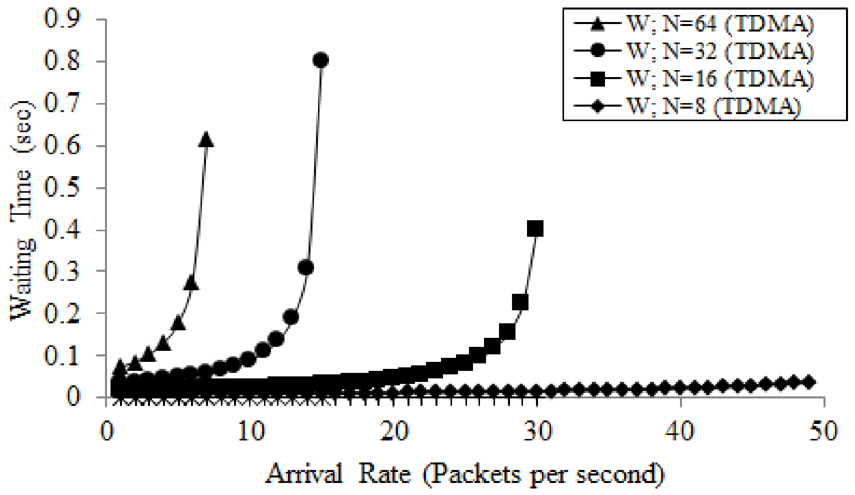


Figure 30(b): Waiting time vs Arrival rate for Poisson based arrivals (TDMA)

The trade-off is between the number of ONUs that can connected for a particular scenario and acceptable delay. For a large number of connected ONUs delay tolerances would have to be relaxed and for strict delay considerations, number of ONUs must be very less.

Following figures show the same models with Pareto distributed arrivals. Figures 31 (a), (b) and (c) show the results based on $x_m = [1,2,3]$, $1 < \alpha < 2$, and $N = [16,32,64]$. It can be seen that as the offered load increases the waiting time increases and after a cetrain threshold the waiting time increases exponentially. This point defines the limit of maximum load that can be supported by the network with a certain number of ONUs.

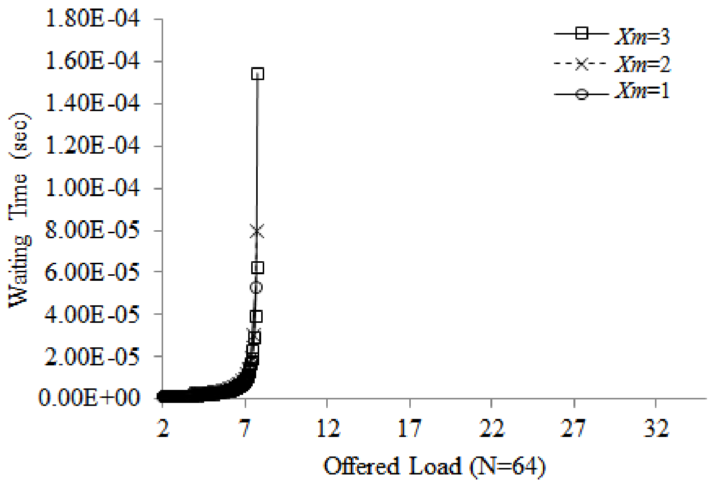


Figure 31(a): Waiting time vs Offered load for (N=64) Pareto arrivals (RC-DBA)

With a decrease in the number of connected ONUs, more traffic load can be afforded in the network.

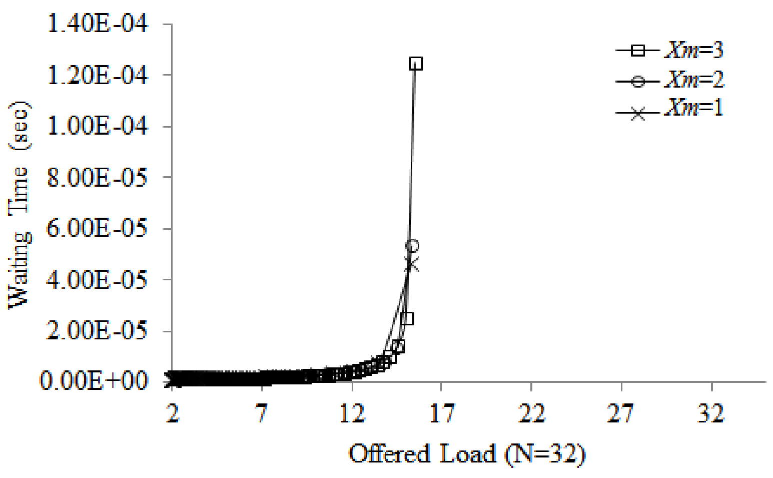


Figure 31(b): Waiting time vs Offered load for (N=32) Pareto arrivals (RC-DBA)

RC-DBA sustains its better performance by keeping the waiting times low.

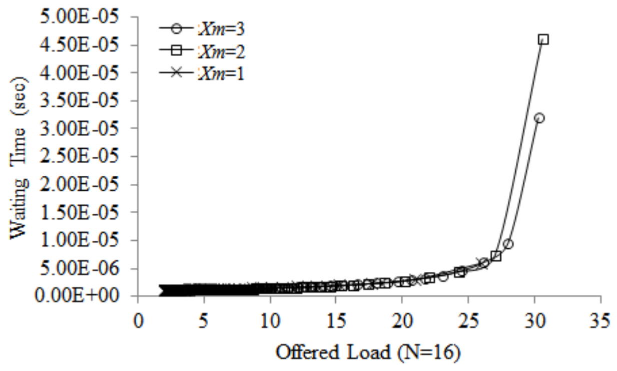


Figure 31(c): Waiting time vs Offered load for (N=16) Pareto arrivals (RC-DBA)

Similarly, in terms of scalability, we can see that delays tend to increase rapidly with as number of ONUs increase. This exponential behavior is prominent in Figures 31(a – c). Since, this is a TDM based solution; there is a direct dependence of average system delay on number of contending users. For SBA, based on the analytical results achieved in previous section (i.e., Eqns. 11 – 14) we show that in terms of latency performance

SBA performs better than the typical requirements of high performance gaming systems. SBA achieves these results for a fan-out of 400+ simultaneous users according to requirements mentioned in chapter 1. Figure 32 shows a comparative view of residual time and waiting time as number of active user's increases.

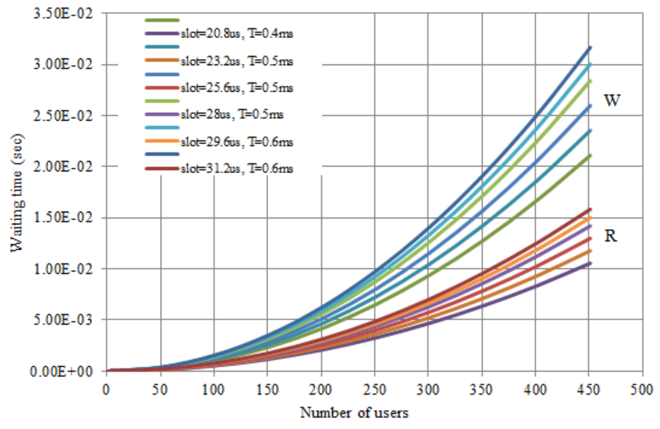


Figure 32: Comparison of waiting time and residual time

For a typical online video gaming system such as OnLive, latency of around 150ms to 200ms [59] can be observed and SBA achieves around 30ms for a cycle time of 0.6ms for 450 users. Similarly, with a relatively smaller cycle time of 0.4ms, SBA achieves 20ms which is still under the acceptable range for OnLive. Since waiting time is a function of average residual times for each user it can be seen in Figure 32 that residual times also show similar behavior.

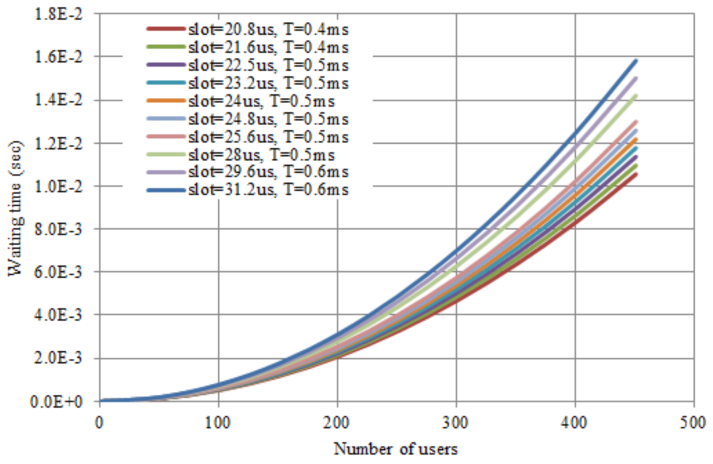


Figure 33: Waiting time increases with increase in number of active users. Various slot times and corresponding cycle times are plotted for a better view of system latency

We realize that with an increase in frame rate from 30fps to 60fps latency increases further but that is to be expected in such a system. With a decrease in number of active users latency increases and can compensate for high frame rate games. Figure 33 shows increase in waiting times with an increase in number of users and as the system reaches around 500 users.

First allocation is where each user gets to transmit regardless of its weight although a user with high weight gets a larger chunk of time. Figure 34 shows waiting time dependence on increasing number of users and increase in arrival rate. With an increase in arrival rate the system tries to expand to accommodate the increased traffic. As a

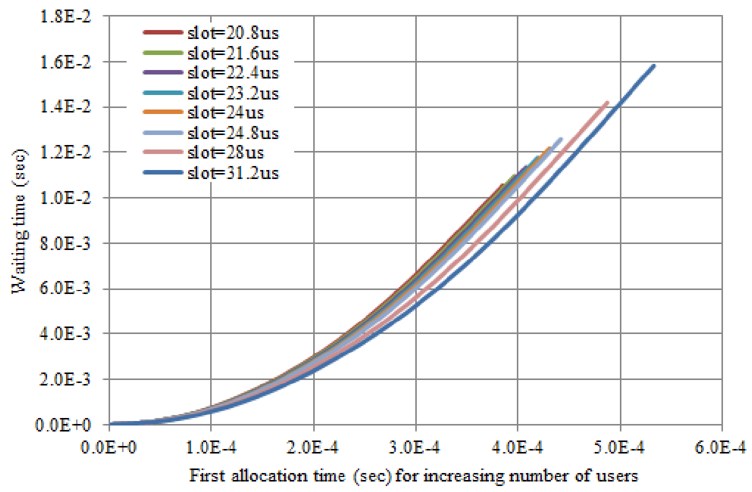


Figure 34: First allocation time dependence on number of users and associated waiting time. Note that this only shows the first phase in bandwidth allocation which is done by allocating based on user weightage

result, latency increases. Moreover, with an increase in cycle time, waiting time also increases and this can be seen in Figure 18, for a range for cycle time values.

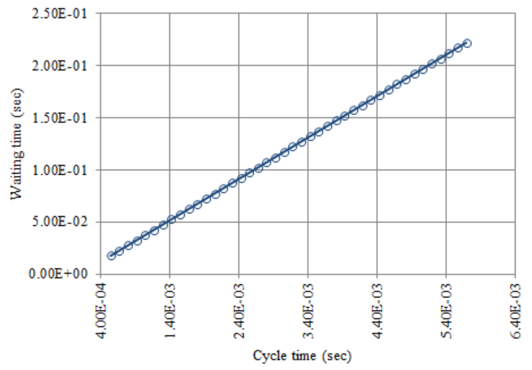


Figure 35: Waiting time increases with increase in cycle time (U = 500)

Presented results clearly show that SBA performs better than existing systems for similar kind of traffic. Also, SBA is novel because it caters for a high-fan-out.

4.5 Discussion

In this chapter we proposed analytical models for latency analysis of RC-DBA and a novel allocation, SBA. Both allocation schemes target different types of networks. While

RC-DBA deals with low-fan-out conventional PONs, SBA allocates resources in high-performance; high-fan-out networks that require SLA based allocation. These schemes are best suited for their respective domains as shown by the results. Performance of SBA can be termed as better than the current day requirements for in-flight video gaming systems, therefore according to the requirements set in chapter 1, SBA is suitable for in-flight implementation. Both RC-DBA and SBA eliminate starvation from the network, in that; each user gets to transmit a minimum time irrespective of SLA. SBA shows better performance in most cases where latency is less than the typical. This helps in increased frame rates for high performance games and also enhances quality of game play experience for regular games of up to 30fps. Latency is less for CSR based systems when compared with SSR based systems although over systems' latency performance is deemed acceptable for lag free gameplay.

The modular nature of SBA gives administrators more leverage, in that; it can also be used with multi-carrier OFDM systems discussed in chapter 2 hence making SBA a useful techniques for multiple architectures. Minimal changes are required for this transition. Instead of single channel, multiple sub-carriers can be treated, each as a single channel. Results presented in this chapter are encouraging since they can be used to unlock further potential in the domain of high-fan-out resource allocation in future works.

CHAPTER 5: DISCUSSIONS

This chapter concludes the dissertation project report by summarizing the project description, methodology, approach and the results. Each chapter presents the results achieved during the various phases of the investigations. Future work has also been identified in the later part of the chapter.

5.1 Concluding remarks

The first phase of the project dealt with the development and verification of a novel high-fan-out short-range optical network which is based on the passive optical networks (PON) architecture on the physical layer. The network layer architecture acts as the backbone for high-fan-out applications discussed in this dissertation and in section 5.2. PONs have now been well established in various renditions over the past decade. We considered the TDM-PON for adaption to enable an in-flight entertainment (IFE) system. In the second phase and at the transport layer a novel real-time video game stream (RVGS) protocol was presented that is sensitive to the video game streams (both CSR and SSR) unlike any other protocol today. In the third phase of the project a novel bandwidth allocation algorithm (SBA) is presented that is state-full in nature. It allocates bandwidth based on the previous states of the system and it is shown that latency performance of SBA is in accordance and in fact better than those of existing online video game systems.

In the contemporary world IFE systems play an important role in enhancing passenger experience. IFE has grown to be a huge industry and companies such as Panasonic and Thales as major shareholders. IFE systems are constrained in their ability to perform, due to inadequate support infrastructure. Entertainment industry and especially the video game industry have evolved rapidly over the years but the IFE systems have remained slow, perhaps due to an evidently focused market. In this project we endeavored to design and test/simulate both the physical layer and the network layer architecture with the help of real-life data.

The project was initiated by identifying the requirements for the networks in constrained environments. At the physical layer level, network architecture has been proposed and investigated that can support high-fan-out applications with high-capacity. The physical layer infrastructure formed the basis and helped develop upper layer models for optimizing full system performance. The designed and demonstrated architecture is best suited for high-capacity bandwidth hungry applications such as in-flight video gaming systems. It is able to provide approximately 200Mb/s to each user with BER kept under the threshold with forward error correction (FEC). It is understood that this bandwidth is much more than the current requirement as evidenced in chapter 3, however keeping in mind the future requirements and the fact that airliners have a life span of few decades, a future proof system was one of the requirements.

Latency models for ratio-counter based dynamic bandwidth allocation (RC-DBA) and analytical and latency model for state-full bandwidth allocation (SBA) technique have been proposed and investigated. Both of the above mentioned allocation techniques target different types of networks. The distinction is drawn based on the number of users each of the networks can serve. RC-DBA allocates bandwidth in conventional low-fan-out optical networks i.e., passive optical networks, whereas SBA is a state-full bandwidth allocation technique that is investigated for allocation of bandwidth in high-fan-out networks. SBA is able to allocate based on priority that can be hard-coded by the system administrator. Otherwise, the priority is set by the system on run time based on demands made by a particular user. In either case, it is ensured that none of the users is starved for bandwidth.

Results show that SBA performs well in terms of latency performance for both client side rendering (CSR) and server side rendering (SSR) based video games. It was shown that approximately 30ms of inter-cycle transmission delay would be incurred by a user on average which is believed to be within the tolerances of around 300ms for online video games. SBA can support the full load of users for upstream transmission as well as downstream transmission which is broadcast. The support extends for games with frame rates in excess of 40fps. Not only in single channel systems, SBA can be used in multiple channel systems such as the QAM OFDM based architecture presented in chapter 2. This shows the flexibility in the technique and adaptability to multiple physical layer scenarios.

5.2 Future work

As mentioned in Chapter 1, there are many applications that can benefit from the work reported in this dissertation. High-fan-out high-capacity systems have many applications besides the IFE. There exist other constrained environments that pose unique challenges in terms of system design and resource allocation. There are a multitude of applications that can benefit from the physical layer architecture and bandwidth allocation techniques demonstrated in this dissertation and those investigations can be done in the future. Thus the proposed system can be optimized for other scenarios based on unique traffic models and physical layer constraints. Some of these applications are pictorially represented in Figure 36 and described below:

1. Train Area Networks (TAN): Trains are becoming increasingly advanced not only in their speed but also in the technology presence on-board. Similar to the in-flight entertainment scenario, train area networks can utilize high-fan-out networks for affording a large number of passengers with large bandwidth.

2. Video gaming competitions (direct application): Perhaps the most direct application of our work is in video gaming competitions. Although, video gaming competitions do not have energy constraints, number of players is large and latency requirements are similar for state-of-the-art video games.

3. Secure networks such that of security agencies that require high-capacity and security: Government agencies such as NASA have a requirement of high-fan-out high-capacity networks due to the nature of their work. Traffic is generated, among other things by high resolution images that are constantly transmitted between terminals. Due to the time critical nature of the job, low latency high-capacity optical network, such as the one proposed in this dissertation, will be able to fulfill the requirements of such a demanding environment.

4. Scientific expositions: Expositions take place all the time and mostly have a large number of participants. A large percentage of the participants are from the industry and academia that showcase the advancements in science and technology. High-speed connectivity among terminals at the expos and with the back end functions is an essential

part of a successful demonstration. A high-speed networks with support for a large number of users is most suitable for such a scenario.

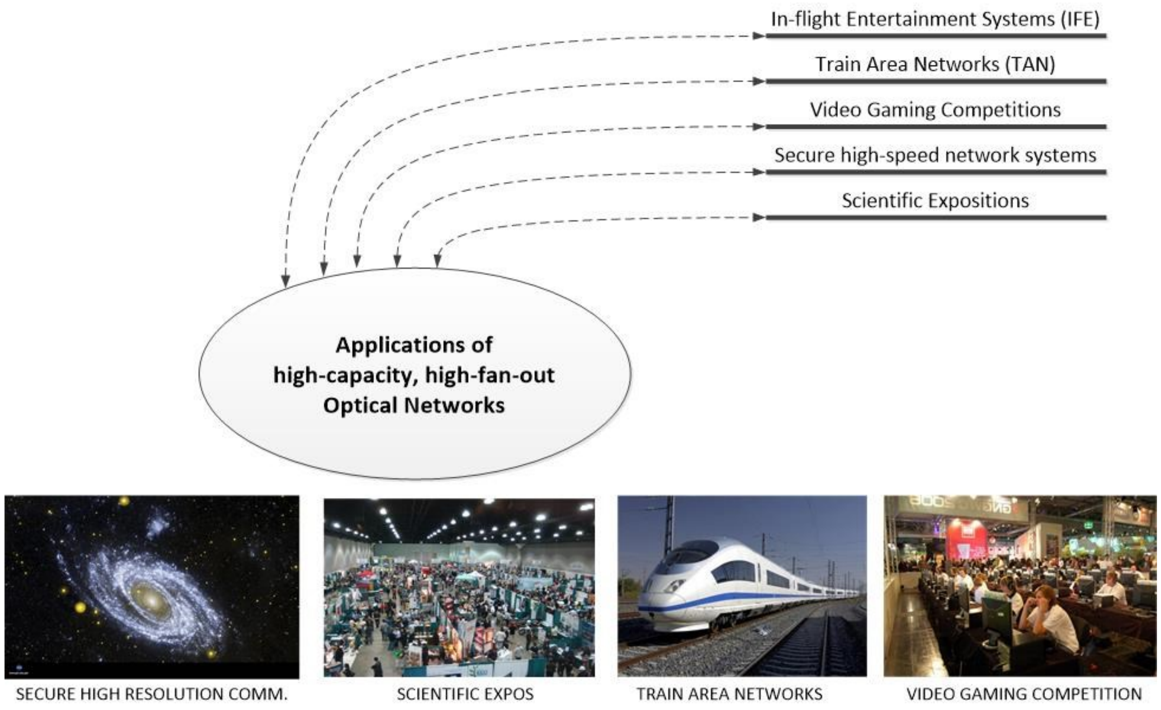

In-flight Entertainment Systems (IFE)
Train Area Networks (TAN)
Video Gaming Competitions
Secure high-speed network systems
Scientific Expositions
Applications of high-capacity, high-fan-out Optical Networks
SECURE HIGH RESOLUTION COMM.
SCIENTIFIC EXPOS
TRAIN AREA NETWORKS
VIDEO GAMING COMPETITION

Figure 36: Applications of high-capacity high-fan-out optical networks

Scenarios mentioned above are just a few of the many that can be enabled with the help of high-speed networks that are tailored to be service oriented. New services are emerging at a high rate and service oriented networks are going to define the efficiency and productivity in the future or optical networking.

REFERENCES

[1] "OnLive - Play on-demand video games over the internet," http://www.onlive.com.

[2] Z. Whittaker, “Electronic Arts: Digital sales will surpass boxed game sales in years,” news.cnet.com, 2012.

[3] B. o. T. Statistics. "June 2011 Airline System Traffic Up 1.6 Percent from June 2010," http://www.bts.gov/press_releases/2011/bts046_11/html/bts046_11.html

[4] S. Perrin, “Cisco 100G Test With EANTC : Overview & Analysis Prepared by,” Cisco, 2012.

[5] M. Birk, P. Gerard, R. Curto et al., “Real-Time Single-Carrier Coherent 100 Gb/s PM-QPSK Field Trial,” Journal of Lightwave Technology, vol. 29, pp. 417-425, 2011.

[6] C. W. M. Robert Craig, Ulrich Sigmund, Video game system using pre-generated motion vectors, USA, USPTO, 2007.

[7] A. Haider, and M. Y. A. Raja, "Latency Analysis of Ratio-Counter based Dynamic Bandwidth," ISCC, 2012, pp. 161 - 165.

[8] http://www.caida.org/home/. "The cooperative association for internet data analysis," 2010,2011,2012.

[9] B. A. Subramanian, In-flight Entertainment, Wipro, 2002.

[10] A. Aviation. "In-Flight Entertainment," http://www.asianaviation.com/articles/160/In-Flight-Entertainment.

[11] Thales. "Innovating for In-flight Entertainment (IFE)," http://www.thalesraytheon-fr.com/Case_Studies/Markets/Aerospace/Innovating_for_inflight_entertainment_(IFE)/.

[12] P. A. Corporation. "Connecting the Business and Pleasure of Flying," http://www.thefutureofifec.com/.

[13] Bigestplane.com, "Seat plan for boeing 777," 2012.

[14] N. A. a. R. Administration., "Code of Federal Regulations," Title 14: Aeronautics and Space.

[15] J. Baliga, R. Ayre, W. V. Sorin et al., “Energy consumption in access networks,” in OFC/NFOEC, 2008.

[16] I. Kang, X. Liu, S. Chandrasekhar et al., "Energy-efficient 0.26-Tb/s coherent-optical OFDM transmission using photonic-integrated all-optical discrete Fourier transform," Optics express/OSA, 2012, pp. 896-904.

[17] S. Wong, L. Valcarenghi, S. Yen et al., “Sleep mode for energy saving PONs: advantages and drawbacks,” 2009.

[18] Y. Zhang, P. Chowdhury, M. Tornatore et al., “Energy efficiency in telecom optical networks,” IEEE Comm. Surverys and Tutorials, 2010.

[19] Z. J. h. Mohamed Elhawary, “Energy-Efficient Protocol for Cooperative Networks,” IEEE/ACM Transactions on Networking, vol. 19, no. 2, pp. 561-574, 2011.

[20] P. Chowdhury, M. Tornatore, A. Nag et al., “On the Design of Energy-Efficient Mixed-Line-Rate (MLR) Optical Networks,” J. Lightwave Technol., vol. 30, no. 1, pp. 130-139, 2012.

[21] F. B. B. Bernhard Schrenk, Johan Bauwelinck, and J. A. L. Josep Prat, “Energy-Efficient Optical Access Networks Supported by a Noise-Powered Extender Box,” IEEE JOURNAL OF SELECTED TOPICS IN QUANTUM ELECTRONICS, vol. 17, no. 2, pp. 480-488, 2011.

[22] L. Shi, B. Mukherjee, and S.-S. Lee, “Energy-Efficient PON with Sleep-Mode ONU: Progress, Challenges, and Solutions,” IEEE Network, pp. 36-41, 2012.

[23] C. P. Lai, A. Naughton, P. Ossieur et al., “Low-Power Colourless Reflective Components for Energy-Efficient Optical Networks,” in ICTON, 2012.

[24] A. Haider, K. Acharya, and M. Y. A. Raja, “Cross-Layer TCP Flow Aware Model for Energy Efficient Use of FP Lasers in Passive Optical Networks,” in HONET, 2010.

[25] W. Li, J. Shao, X. Liang et al., “An optical OFDM multiplexer and 16×10Gb/s OOFDM system using serrodyne optical frequency translation based on LiNbO3 phase modulator,” Optics Communications, vol. 284, pp. 3970-3976, 2011.

[26] R. Nagarajan, J. Rahn, M. Kato et al., “10 Channel, 45.6 Gb/s per Channel, Polarization-Multiplexed DQPSK, InP Receiver Photonic Integrated Circuit,” Journal of Lightwave Technology, vol. 29, pp. 386-395, 2011.

[27] W. Shieh, H. Bao, and Y. Tang, “Coherent optical OFDM: theory and design.,” Optics express, vol. 16, pp. 841-59, 2008.

[28] J. Yu, M.-f. Huang, and D. Qian, “Centralized Lightwave WDM-PON Employing 16-QAM Intensity Modulated OFDM Downstream and OOK Modulated Upstream Signals,” Photonic Technology Letters, vol. 20, pp. 1545-1547, 2008.

[29] D. Foursa, Y. Cai, J. Cai et al., “Coherent 40 Gb / s Transmission with High Spectral Efficiency Over Transpacific Distance,” OFC, pp. 4-6, 2011.

[30] B. Zhao, and X. Chen, “A 40Gbps SSB-OFDMA-PON Architecture Using Direct-Detection and Source-Free ONUs Supporting Dynamic Bandwidth Allocation,” 2011 Third International Conference on Communications and Mobile Computing, pp. 223-225, 2011.

[31] C. E. Shannon, “A Mathematical Theory of Communication,” Bell System Technical Journal, vol. 27, no. 3, pp. 379-423, 1948.

[32] “Ciena - Ciena Widens Leadership in High Speed Optics with Innovative WaveLogic 3 Technology - WaveLogic 3, 100G, Coherent, 400 Gb/s, programmable.”

[33] “The 400G Photonic Service Engine | TechZine - Alcatel-Lucent.”

[34] FFT Tutorial, University of Rhode Island, Department of Electrical Engineering and Computer Science.

[35] G. Bosco, A. Carena, V. Curri et al., “Performance Limits of Nyquist-WDM and CO-OFDM in High-Speed PM-QPSK Systems,” Photonic Technology Letters, vol. 22, pp. 1129-1131, 2010.

[36] G. Shen, and M. Zukerman, “Spectrum-efficient and agile CO-OFDM optical transport networks: architecture, design, and operation,” IEEE Communications Magazine, vol. 50, pp. 82-89, 2012.

[37] X. Wang, and G. B. Giannakis, “Resource Allocation for Wireless Multiuser OFDM Networks,” IEEE Transactions on Information Theory, vol. 57, pp. 4359-4372, 2011.

[38] D. Johnson. "Fourier Series Approximation of a Square Wave," 09/09/12, 2012.

[39] H. L. Cooke, Time Division Multiplex System for Signals of Different Bandwidth, 2,919,308, U. P. Office, 1954.

[40] W. Shieh, “OFDM for Flexible High-Speed Optical Networks,” Journal of Lightwave Technology, vol. 29, pp. 1560-1577, 2011.

[41] D. I. Program, "Internet Protocol DARPA Internet Program Protocol Specificncation," Defence Advanced Research Projects Agency, 1981.

[42] "Ethernet Frame," Sep, 2012; http://www.infocellar.com/networks/ethernet/frame.htm.

[43] J. F. Kurose, and K. W. Ross, Computer Networking: A Top-Down Approach, 6 ed.: Addision Wesley, 2012.

[44] B. Anand, J. Sebastian, S. Y. Ming et al., “PGTP: Power aware game transport protocol for multi-player mobile games,” in Communications and Signal Processing (ICCSP), 2011 International Conference on, 2011, pp. 399-404.

[45] S. Pack, E. Hong, Y. Choi et al., “Game Transport Protocol: A Reliable Lightweight Transport Protocol for Massively Multiplayer On-line Games (MMPOGs),” 2002.

[46] U. K. Mufti, S. A. Haider, and S. M. H. Zaidi, "Ratio-counter based dynamic bandwidth allocation algorithm (RCDBA) extending EFDBA," HONET 2009, IEEE, 2009, pp. 115-119.

[47] IEEE, "Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications," Physical Layer Specifications and Management Parameters for 10Gb/s Passive Optical Networks, IEEE, 2009.

[48] F. Wang, and C. Chen, “Dynamic bandwidth allocation algorithm based on idle times over Ethernet PONs,” in ICSPCC, 2011.

[49] M. Tanaka, T. Nishitani, H. Mukai et al., “Adaptive dynamic bandwidth allocation scheme for multiple-services in 10GEPON systems,” in ICC, 2011.

[50] B. Skubic, J. Chen, J. Ahmed et al., “Dynamic bandwidth allocation for long-reach PON: overcoming performance degradation,” IEEE Communications Magazine, vol. 48, no. 11, November 2010, 2010.

[51] K. Christodoulopoulos, I. Tomkos, and E. A. Varvarigos, “Routing and Spectrum Allocation in OFDM-Based Optical Networks with Elastic Bandwidth Allocation ” in GLOBECOM, 2010, pp. 1-6.

[52] B. Park, A. Hwang, and J. H. Yoo, “Enhanced Dynamic Bandwidth Allocation Algorithm in EPONs,” ETRI Journal, vol. 30, no. 2, 2008.

[53] A. Gumaste, K. Pulverer, A. Teixeira et al., “Medium access control for the next-generation passive optical networks: the OLIMAC approach,” IEEE Network, vol. 26, pp. 49-56, 2012.

[54] S. Kawanishi, H. M. Yokosuka Takara, T., O. Kamatani et al., “Single channel 400 Gbit/s time-division-multiplexed transmission of 0.98 ps pulses over 40 km employing dispersion slope compensation,” Electronics Letters, vol. 32, pp. 916-918, 2002.

[55] "Cisco Systems' 100 Gigabit spans metro to ultra long-haul," May 2012, 2012; http://www.gazettabyte.com/home/2012/3/7/cisco-systems-100-gigabit-spans-metro-to-ultra-long-.

[56] "ParetoDistribution," Sep, 2012; http://reference.wolfram.com/mathematica/ref/ParetoDistribution.html.

[57] M. O. Lorenz, Methods of Measuring the Concentration of Wealth, p.^pp. 209-219: Publications of the American Statistical Association, 1905.

[58] L. Kleinrock, Queueing Systems, Volume 1, Theory: John Wiley & Sons, 1975.

[59] R. Leadbetter, “Console Gaming: The Lag Factor,” Eurogamer.net, 2009.

APPENDIX A: RC-DBA CODE

```
#include <iostream>
#include <math.h>
#include <fstream>
#include "rngs.h"
#include "rngs.c"

#define N 8
#define datarate 1E+09
#define PI 3.147

using namespace std;

int populateRequests(int []);
float generatePoisson();
void printList(int []);
void printList(double []);
int convertBytesToTime(int [], double []);
int convertTimeToBytes(double [], int []);
int calResidualTimes(double [],double [],double,int []);
double compressEnsuredWindow(double [],int [],double);
double findMinRequest(double [],int []);
int calUnitsAllocated(int [],double [],double,int [],int);
int allocateRestWindow(int [],double [],int [],double);
int calResidualTimeForNextSlot(int [],double [],double [],double,double []);
int printChart(int [],int [],int [],int [],int []);

int main (int argc, char *argv[])
{
    int slot=1, unitsAllowedInRest;
    int ratioCounter[N], reqBytes[N],unitsAllocated[N], residualBytesForNextSlot[N],
restWindowAllocated_Bytes[N];
    double reqTime[N], residualTime[N], restWindowAllocated[N],
residualTimeForNextSlot[N];
    double T=0.002,T_G=0.000001,oneSlot,compressedWindowSize,minOverReq;
    char quit;

    oneSlot=(T/2)/N;
    for (int K=0;K<N;K++) ratioCounter[K]=0;
    for (int K=0;K<N;K++) unitsAllocated[K]=0;
    populateRequests(reqBytes);     //populate the request list
    //printList(reqBytes);          //print the passed list
    convertBytesToTime(reqBytes,reqTime);     //printList(reqTime);
    calResidualTimes(reqTime,residualTime,oneSlot,ratioCounter);   //Ensured window -
requested time
    //printList(residualTime);
    //printList(ratioCounter);
    compressedWindowSize=compressEnsuredWindow(reqTime,ratioCounter,oneSlot);
    cout<<endl<<"TOTAL="<<T<<" Ensured="<<compressedWindowSize<<" Rest="<<T-
compressedWindowSize<<endl;
    minOverReq=findMinRequest(residualTime,ratioCounter);

    unitsAllowedInRest = (T-compressedWindowSize)/minOverReq;

    cout<<endl<<"MINIMUM REQUEST: "<<minOverReq<<endl;
    calUnitsAllocated(unitsAllocated,residualTime,minOverReq,ratioCounter,
unitsAllowedInRest);
    //printList(unitsAllocated);
    allocateRestWindow(ratioCounter,restWindowAllocated,unitsAllocated,minOverReq);
    convertTimeToBytes(restWindowAllocated,restWindowAllocated_Bytes);
    //printList(restWindowAllocated);

calResidualTimeForNextSlot(ratioCounter,residualTimeForNextSlot,reqTime,oneSlot,restWindo
wAllocated);
    //printList(residualTimeForNextSlot);
    convertTimeToBytes(residualTimeForNextSlot,residualBytesForNextSlot);
    //printList(residualBytesForNextSlot);
printChart(reqBytes,ratioCounter,unitsAllocated,restWindowAllocated_Bytes,residualBytesFo
rNextSlot);

    //convertTimeToBytes()
     quit = '\0';
    while (quit != 'q')
    {
        cout << "Press q to quit " << endl;
        cin >> quit;
    }
    return 0;
}
int populateRequests(int reqBytes[])
{

   for (int i=0; i<N; i++)
        {
        reqBytes[i]= generatePoisson();
        }


    reqBytes[0]=100000;

    return 1;
}

float generatePoisson()
{

    float lambda = 10,u,x,n,v,y,lhs,rhs;
    float c = 0.767 - 3.36/lambda;
    float beta = PI/sqrt(3.0*lambda);
    float alpha = beta*lambda;
    float k = log(c) - lambda - log(beta);

    while(1)
    {
       u = rand();
       x = (alpha - log((1.0 - u)/u))/beta;
       n = floor(x + 0.5);
       if (n < 0)
               {
               v = rand();
               y = alpha - beta*x;
               lhs = y + log(v/(1.0 + exp(y))^2);
               rhs = k + n*log(lambda) - lgamma(n+1); //log(n!)
               if (lhs <= rhs)
                       return n;
          }
      }

}

void printList(int inputList[])
{
     for (int i=0; i<N; i++)
        {
        cout<<endl<<"ONU-"<<i<<"-->"<<inputList[i]<<endl;
        }

}
void printList(double inputList[])
{
     for (int i=0; i<N; i++)
        {
        cout<<endl<<"ONU-"<<i<<"-->"<<inputList[i]<<endl;
        }

}
```

```
int convertBytesToTime(int Bytes[], double Time[])
{
    for (int i=0; i<N; i++)
        {
        Time[i]=Bytes[i]/datarate;
        }
    return 1;
}

int convertTimeToBytes(double Time[], int Bytes[])
{
    for (int i=0; i<N; i++)
        {
        Bytes[i]=Time[i]*datarate;
        }
    return 1;
}

int calResidualTimes(double Request[],double Residual[],double oneSlotTime,int Counter[])
{
    for (int i=0; i<N; i++)
        {
        Residual[i]=oneSlotTime-Request[i];
           if (Residual[i] < 0) Counter[i]=1; else Counter[i]=0;               //set
ratio counter here as well
        }
    return 1;

}

double compressEnsuredWindow(double Request[], int Counter[],double oneSlotSize)
{
double size=0;
    for (int i=0; i<N; i++)
        {
         if (Counter[i]==0) size+=Request[i];
         else if (Counter[i]==1) size+=oneSlotSize;
        }
    return size;
}

double findMinRequest(double Residual[],int Counter[])
{
double minReq=999999;
    for (int i=0;i<N;i++)
        {
        if ((Counter[i]==1) && (Residual[i]*-1 < minReq)) minReq = Residual[i]*-1;
        }
    return minReq;
}

int calUnitsAllocated(int Units[],double Residual[],double minReq, int Counter[],  int
maxUnitsAllowed)
{
int k=0;
while (k<maxUnitsAllowed)
      {
      for(int i=0;i<N;i++)
         {
         if(Counter[i]==1)
            {
            Units[i]=Units[i]+1;
            k++;
            if (Units[i] == Residual[i]*-1/ minReq) Counter[i]=0;
            }
         else
             Units[i]=0;

         }
      }

/*

   for (int i=0;i<N;i++)
        {
        if (Counter[i]==1)
           Units[i]=Residual[i]*-1/ minReq;
        else
           Units[i]=0;
        }
*/
   return 1;
}

int allocateRestWindow(int Counter[],double restAllocated[],int Units[],double minReq)
{
    for (int i=0;i<N;i++)
        {
        if (Counter[i]==1)
           restAllocated[i] = Units[i] * minReq;
        else
           restAllocated[i]=0;
        }
   return 1;
}

int calResidualTimeForNextSlot(int Counter[],double residualTimeNextSlot[],double
reqTime[],double slotSize, double restAllocated[])
{
    for (int i=0;i<N;i++)
        {
        if (Counter[i]==1) residualTimeNextSlot[i]=reqTime[i]-slotSize-restAllocated[i];
        else residualTimeNextSlot[i]=0;
        }
   return 1;
}

int printChart(int requestedBytes[],int Counter[],int Units[],int
restAllocated_Bytes[],int residualBytes[])
{
cout<<"Request\t\t"<<"Counter\t"<<"Units\t"<<"Rest\t"<<"Residual"<<endl;

    for(int i=0;i<N;i++)
       {

cout<<requestedBytes[i]<<"\t\t"<<Counter[i]<<"\t"<<Units[i]<<"\t"<<restAllocated_Bytes[i]
<<"\t"<<residualBytes[i]<<endl;
       }
    return 1;
}
```

APPENDIX B: RC-DBA & EFDBA WAITING TIMES CODE

```
#include <iostream>
#include <math.h>
#include <fstream>

using namespace std;

int main (int argc, char *argv[])
{
    int N=16,n=1,i,j,lambda=557, alpha=2,x_m=1;
    double T=0.002,T_G=0.000001,L_S,W,R,Row=0.1,E_Pareto;
    L_S = T/(2*N);
    char quit;
    //open file for writing
    ofstream myfile;
    myfile.open ("Waiting_Time_High.txt");


//for(double T=0.001; T<0.005; T=T+0.001)
//        {
 //        cout << endl << "t=" << T << endl;
 //        myfile << endl << "t=" << T << endl;
//         for(int lambda=1;lambda<50;lambda++)
for (x_m=1;x_m<10;x_m++)
    {
         for(float alpha=1.0;alpha<2.0;alpha=alpha+0.01)

         //while (Row < 0.9)
                 {
                 E_Pareto=(alpha * x_m)/(alpha-1);
                 R=T*((3/4)-(1/(N*n)))+T_G;
//                 W=R/(1-(n*lambda*N*T));
                 W=R/(1-(n*E_Pareto*N*T));

//                 cout << "N=" << N <<", n="<< n <<", lambda="<< lambda <<", W="<< W <<
endl;
                 cout << "x_m="<<x_m<<", N=" << N <<", n="<< n <<", alpha="<< alpha <<",
E_Pareto="<< E_Pareto<<", W="<< W << endl;

                 myfile << "x_m="<<x_m<<", N=" << N <<", n="<< n <<", alpha="<< alpha<<",
E_Pareto="<< E_Pareto <<", W="<< W << endl;
                 Row=Row+.1;
                 W=0.0;R=0.0;
                 //N=N+1;
                 }
      }
//         }
//}
    quit = '\0';
    while (quit != 'q')
    {
        cout << "Press q to quit " << endl;
        cin >> quit;
    }
    myfile.close();
    return 0;
}
```

```
#include <iostream>
#include <math.h>
#include <fstream>

using namespace std;

int main (int argc, char *argv[])
{
    int N=64,n=1,i,j,lambda=557,pkt_len=30*8;
    double T=0.002,T_G=0.000001,L_S,W,R,Row=0.1,data_rate=1E09;
    L_S = T/(2*N);
    char quit;
    //open file for writing
    ofstream myfile;
    myfile.open ("Waiting_Time_EFDBA.txt");
for(int N=8; N<65; N=N*2)
{
       cout << endl << "N=" << N << endl;
       myfile << endl << "=,=,=,=W;N=" << N <<" (EFDBA)"<< endl;
         for(int lambda=1;lambda<50;lambda++)
         //while (Row < 0.9)
                 {

                 R= (T*((2*n*(3*N - 2) - 1)/(4*N*n))+T_G);
                 W=R/(2*(1-(n*((lambda*pkt_len)/data_rate)*N*T)));
                 cout << "N=" << N <<", n="<< n <<", lambda="<< lambda <<", W="<< W <<
endl;
                 myfile << "N=" << N <<", n="<< n <<", lambda="<< lambda <<", W="<< W <<
endl;
                 Row=Row+.1;
                 W=0.0;R=0.0;
                 //N=N+1;
                 }
//         }
}
    quit = '\0';
    while (quit != 'q')
    {
        cout << "Press q to quit " << endl;
        cin >> quit;
    }
    myfile.close();
    return 0;
}
```

APPENDIX C: SBA CODE

```
#include <iostream>
#include <math.h>
#include <fstream>
#include <conio.h>

#define N 500
#define C 100 //number of cycles
#define datarate 100E+09
#define pkt_len 805
#define T_G 200E-09
#define cycleSet 10
#define AT_2ND 12 //allocation table 2nd index value

using namespace std;
int populateRequests(int []); //incoming requests
int populateWeightTable(int [][2]); //assign weights
void printList(int []);
void printList(int [][2]);
void printList(double []);
void printList(double [][AT_2ND]);
void printResidual(double [][C]);
int convertBytesToTime(int [], double []);
//int convertTimeToBytes(double [], int []);
int populateAllocTable(double [][AT_2ND],int,double []);
int assignFA(double [][AT_2ND], int [][2], int, double, double, double [],double
[][C],int); //alloc_table,weight_table,units,slot_time,cycle_time,req_time,residual_time
int printInFile(double [][AT_2ND]);
int assignSA(double [][AT_2ND], int [][2], int, double, double, double [],double
[][C],int);
//alloc_table,weight_table,units,slot_time,cycle_time,req_time,residual_time,cycle
int printResidualInFile(double [][C]);

int main (int argc, char *argv[])
{
    int units,cycle=0;
    int weightTable[N][2]; //(#,w)
    double
allocTable[N][AT_2ND];//(#,A?,w,cycle,req_current,max_current,min_current,allocated_curre
nt,SAbit?,residual,allocated_SA,left_bytes_for_next_cycle
    int reqBytes[N];
    double reqTime[N],residualTime[N][C];
    //int ratioCounter[N], unitsAllocated[N], residualBytesForNextSlot[N],
restWindowAllocated_Bytes[N];
    //double residualTime[N], restWindowAllocated[N], residualTimeForNextSlot[N];
    double T,slotTime;//,compressedWindowSize,minOverReq;
    char quit;

    //open file for writing
    ofstream myfile;
    myfile.open ("S-BA.txt");

    populateWeightTable(weightTable); // assign weights
    //printList(weightTable);

    //INITIALIZE ALLOCATION TABLE
    for(int i=0;i<N;i++) for(int j=0;j<AT_2ND;j++) allocTable[i][j]=0;
    //INITIALIZE RESIDUAL TIME
    for(int i=0;i<N;i++) for(int j=0;j<C;j++) residualTime[i][j]=0;


for(int c=0; c<C; c++)
{

    //CALCULATE SLOT-TIME
    slotTime=((pkt_len*8)/datarate) + T_G; //slot time equal to time it takes to transmit
one packet
    //calculate T_C (Cycle Time)
    units=0;
    for(int i=0;i<N;i++) units += weightTable[i][1];
    T=(slotTime * units);
    T += T/2;
    //cout<<T;getch();
    //CALCULATE SLOT-TIME (END)
    populateRequests(reqBytes);
    convertBytesToTime(reqBytes,reqTime);

    for(int i=0; i<N; i++)
       {
       allocTable[i][4]=allocTable[i][4]+allocTable[i][11];
       allocTable[i][11]=0.0;
       }

    //ASSIGN First Allocation
    assignFA(allocTable,weightTable,units,slotTime,T,reqTime,residualTime,cycle);
    assignSA(allocTable,weightTable,units,slotTime,T,reqTime,residualTime,cycle);
    //printList(allocTable);
    printInFile(allocTable);
    cycle++;     //increment cycle
cout<<"Cycle: "<<c<<endl;

}
    //printResidual(residualTime);
    printResidualInFile(residualTime);

    quit = '\0';
    while (quit != 'q')
    {
        cout << "Press q to quit " << endl;
        cin >> quit;
    }
    return 0;
}


int populateRequests(int reqBytes[])
{
    int alpha=2;
    //INITIALIZE & POPULATE
    for(int j=0;j<N;j++) reqBytes[j]=0;
     for(int j=0;j<N;j++)
        {
        reqBytes[j]=((alpha*(gen())/(alpha-1)); //scaled request space Pareto
        }

    return 1;
}

int populateWeightTable(int weight_table[][2])
{
     //INITIALIZE & POPULATE
    for(int i=0;i<N;i++) for(int j=0;j<2;j++) weight_table[i][j]=0;
     for(int j=0;j<N;j++)
    {
        weight_table[j][0]=j;
        weight_table[j][1]=rand() % 3 + 1;

    }

    return 1;
}

void printList(int inputList[])
{
     for (int i=0; i<N; i++)
        {
        cout<<"ONU-"<<i<<"-->"<<inputList[i]<<endl;
        }

}
```

```
void printList(int inputList[][2])
{
     for (int i=0; i<N; i++)
        {
        cout<<"ONU-"<<i<<"-->"<<inputList[i][1]<<endl;
        }

}
void printList(double inputList[])
{
     for (int i=0; i<N; i++)
        {
        cout<<endl<<"ONU-"<<i<<"-->"<<inputList[i]<<endl;
        }

}

void printList(double inputList[][AT_2ND])
{

//(#,A?,w,cycle,req_current,max_current,min_current,allocated_current,SAbit?,residual,all
ocated_SA,left_bytes_for_next_cycle

cout<<endl<<"#\tA?\tw\tcycle\treq_current\tmax_current\tmin_current\talocated_current\tSA
bit?\tresidual\tallocated_SA\tbytes_carry"<<endl;
     for (int i=0; i<N; i++)
        {
        cout<<endl;
        for (int j=0; j<AT_2ND; j++)
            {
            cout<<inputList[i][j]<<"\t";
            }
         }
}

void printResidual(double inputList[][C])
{
     cout<<endl<<"0\t1\t2\t3\t4\t5\t6\t7\t8\t9"<<endl;
     for (int i=0; i<N; i++)
        {
        cout<<endl;
        for (int j=0; j<C; j++)
            {
            cout<<inputList[i][j]<<"\t";
            }
         }

 }


int convertBytesToTime(int Bytes[], double Time[])
{
    for (int i=0; i<N; i++)
        {
        Time[i]=Bytes[i]/datarate;
        }
    return 1;
}

/*int convertTimeToBytes(double Time[], int Bytes[])
{
    for (int i=0; i<N; i++)
        {
        Bytes[i]=Time[i]*datarate;
        }
    return 1;
}*/

int populateAllocTable(double alloc_table[][AT_2ND],int col,int row,double val[])
{
    for (int i=0; i<N; i++)
```

```
        {
        alloc_table[i][col]=val[i];
        }
    return 1;
}

int assignFA(double alloc_table[][AT_2ND], int weight_table[][2], int unit, double
slot_time, double t, double req_time[], double residual_time[][C], int cyc)
//alloc_table,weight_table,units,slot_time,cycle_time
{
    //weight_table(#,w)

//alloc_table(#,A?,w,cycle,req_current,max_current,min_current,allocated_current,SAbit?,r
esidual,allocated_SA,left_bytes_for_next_cycle
    for(int i=0; i<N; i++)
        {
         alloc_table[i][0] = i;
         alloc_table[i][1] = 1;
         alloc_table[i][2] = weight_table[i][1];
         alloc_table[i][3] = cyc;
         alloc_table[i][4] = req_time[i];
         alloc_table[i][5] = slot_time * weight_table[i][1]; //max allowed in current
         alloc_table[i][6] = slot_time * weight_table[i][1]; //min allowed time

         if(alloc_table[i][4] <= alloc_table[i][5]) //if requested < max allowed
             {
             alloc_table[i][7] = alloc_table[i][4];                 //allocate requested
             //residualTime[i] = alloc_table[i][4]-alloc_table[i][5]; //put rest in
residual (-ve)
             alloc_table[i][8]= 0;
             alloc_table[i][9] = alloc_table[i][4]-alloc_table[i][5]; //put rest in
residual (-ve)
             }
         else if (alloc_table[i][4] > alloc_table[i][5]) //if requested > max allowed
             {
             alloc_table[i][7] = alloc_table[i][5];      //allocate max allowed
             //residualTime[i] = alloc_table[i][4]-alloc_table[i][5]; //put rest in
residual (+ve)
             alloc_table[i][8]= 1;
             alloc_table[i][9] = alloc_table[i][4]-alloc_table[i][5]; //put rest in
residual (+ve)
             }

         alloc_table[i][10]=0.0;
         alloc_table[i][11]=0.0;
        }
     return 1;

}

int printInFile(double inputList[][AT_2ND])
{

//open file for writing
    ofstream myfile;
    myfile.open ("S-BA.txt");
 myfile
<<endl<<"#\tA?\tw\tcycle\treq_current\tmax_FA\tmin_FA\tallocated_FA\tSAbit?\tresidual(+ve
req)\tallocated_SA\tbytes_carry"<<endl;
     for (int i=0; i<N; i++)
        {
         myfile <<endl;
        for (int j=0; j<AT_2ND; j++)
            {
             myfile <<inputList[i][j]<<"\t";
            }
         }
 myfile.close();
}
```

```
int assignSA(double alloc_table[][AT_2ND], int weight_table[][2], int unit, double
slot_time, double t, double req_time[], double residual_time[][C], int cyc)
//alloc_table,weight_table,units,slot_time,cycle_time
{
    int units=0;
    double time_used_in_FA=0.0,time_left_for_SA=0.0,slot=0.0;
    //weight_table(#,w)

//alloc_table(#,A?,w,cycle,req_current,max_current,min_current,allocated_current,SAbit?,r
esidual,allocated_SA,left_bytes_for_next_cycle
    for(int i=0; i<N; i++)
        {

         if(alloc_table[i][8])
            {
            units+= weight_table[i][1]; //add all units
            }
        time_used_in_FA += alloc_table[i][7];
        }

        //also include other cases
        time_left_for_SA = t - (time_used_in_FA + (N * T_G));

        slot = time_left_for_SA/units;
        //cout<<slot<<endl<<units;getch();


    for(int j=0; j<N; j++)
         {
         if(alloc_table[j][8])
            {

            if (alloc_table[j][9] < (weight_table[j][1] * slot))
               {
               alloc_table[j][10]=alloc_table[j][9];//allocate residual only
               alloc_table[j][11]=0;
               }
            else if (alloc_table[j][9] >= (weight_table[j][1] * slot))
               {
               alloc_table[j][10]=weight_table[j][1] * slot;//allocate max possible in SA
               alloc_table[j][11]=alloc_table[j][9]-alloc_table[j][10];//record what is
still left
               }

            residual_time[j][cyc]=alloc_table[j][11] ;//also populate residual_time for
this cycle
            }

         }
     return 1;

}

int printResidualInFile(double inputList[][C])
{
   //open file for writing
    ofstream myfile;
    myfile.open ("S-BA_residual.txt");
    for (int j=0; j<C; j++) myfile<<j<<"\t";
    cout<<endl;
     for (int i=0; i<N; i++)
        {
         myfile <<endl;
        for (int j=0; j<C; j++)
            {
             myfile <<inputList[i][j]<<"\t";
            }
         }
 myfile.close();
}
```

APPENDIX D: SBA WAITING TIME CODE

```
#include <iostream>
#include <math.h>
#include <fstream>

using namespace std;

int main (int argc, char *argv[])
{

    char quit;
    int N=500,alpha=2,x_m=50,u=5,n;
    int WEIGHT_TABLE[N][2]; //(#,w)
    int MIN_ALLOC[3][2]; //weight,# of users(N_i)
    double ALLOC_TABLE[N][8];//user
#,weight,cycle,request_current_cycle,max_current_cycle,min_current_Cycle,allocated_FA,all
ocated_SA
    double r=100E+09,T_G=200E-09,T,R,W,tau,units,E_Pareto,row,SERV_RATE=12E+09;


    //open file for writing
    ofstream myfile;
    myfile.open ("Waiting_Time_High.txt");

    //INITIALIZE TABLES
    for(int i=0;i<N;i++) for(int j=0;j<2;j++) WEIGHT_TABLE[i][j]=0;
    for(int i=0;i<3;i++) {MIN_ALLOC[i][0]=i+1;MIN_ALLOC[i][1]=0;}
    for(int i=0;i<N;i++) for(int j=0;j<8;j++) ALLOC_TABLE[i][j]=0;

    //Print MIN_ALLOC
    //for(int i=0;i<3;i++) cout<<"\nWeight: "<<MIN_ALLOC[i][0]<<" No.:
"<<MIN_ALLOC[i][1];


    for(int j=0;j<N;j++)
    {
        WEIGHT_TABLE[j][0]=j;
        WEIGHT_TABLE[j][1]=rand() % 3 + 1;
        if(WEIGHT_TABLE[j][1] == 3) MIN_ALLOC[2][1]++;
        else if(WEIGHT_TABLE[j][1] == 2) MIN_ALLOC[1][1]++;
        else if(WEIGHT_TABLE[j][1] == 1) MIN_ALLOC[0][1]++;
    }
    //Print MIN_ALLOC
    //for(int i=0;i<3;i++) cout<<"\nWeight: "<<MIN_ALLOC[i][0]<<" No.:
"<<MIN_ALLOC[i][1];
    //Print WEIGHT_TABLE
    //for(int i=0;i<N;i++) cout<<"\nUser: "<<WEIGHT_TABLE[i][0]<<" w:
"<<WEIGHT_TABLE[i][1];

    double pkt_len,slot_time;//=805;//bytes
for (pkt_len=1000;pkt_len<150000;pkt_len=pkt_len+1000)
{
    //slot_length
    //pkt_len=805;
    slot_time=((pkt_len*8)/r) + T_G; //slot time equal to time it takes to transmit one
packet
    cout<<"\nPKT_LEN: "<<pkt_len<<"bytes\nSLOT_TIME: "<<slot_time;



    cout <<"\n";
//for (x_m=0;x_m<1500;x_m=x_m+100)
//for (alpha=1;alpha<10;alpha++)
//{
//for(n=1;n<N;n=n+50)
n=N;
//for (int u=0;u<N;u++)
//for(T=0.0001;T<0.005;T=T+0.0001)
//    {
```

```
                 //T_C (Cycle Time)
                 units=0;
                 for(int i=0;i<n;i++) units += WEIGHT_TABLE[i][1];
                 T=(slot_time * units);
                 T += T/2;
                 T += T_G * n;
                 cout<<"\nUNITS: "<<units<<"\nT_C: "<<T<<"\n";


                 E_Pareto = ((alpha*x_m)/(alpha-1));
                 row = E_Pareto / SERV_RATE;
                 //w = 550 * pkt_len * 8 / SERV_RATE;
                 cout <<"\nE_Pareto: "<< E_Pareto<<"\n";

                 //Calculate tau
                 //for(int i=0;i<3;i++) units += MIN_ALLOC[i][0] * MIN_ALLOC[i][1];
                 //tau= T / (2*units);
                 tau=slot_time;
                 //cout<<"\nunits: "<<units<<"\n";

                 //Calculate Residual Time
                 //for(int i=0;i<3;i++) R += MIN_ALLOC[i][0] * MIN_ALLOC[i][1] * tau;
                 for(int i=0;i<u-1;i++) R += WEIGHT_TABLE[i][0] * tau;
                 for(int i=u+1;i<n;i++) R += WEIGHT_TABLE[i][0] * tau;

                 //if (E_Pareto > )
                 //R += (n-1)* (E_Pareto/r);
                 //R += T/2;
                 R +=T_G;

                 //Determine Waiting Time
                 W = (1*R) / (2*(1-row));

                 cout << "T="<<T << ", tau="<<tau << ", x_m="<<x_m<<", n=" << n <<",
pkt_len=" << pkt_len <<", alpha="<< alpha <<", R="<< R <<", W="<< W << endl;

                 myfile << "T="<<T<<", tau="<<tau<< ", x_m="<<x_m<<", n=" << n <<",
pkt_len=" << pkt_len <<", alpha="<< alpha<<", R="<< R <<", W="<< W << endl;

                 W=0.0;R=0.0;tau=0.0;units=0.0;row=0.0;E_Pareto=0.0;

     }
//}

//}//outer for loop


    quit = '\0';
    while (quit != 'q')
    {
        cout << "Press q to quit " << endl;
        cin >> quit;
    }
    myfile.close();
    return 0;
}
```

APPENDIX E: DATA CAPTURE CODE

```
#include <stdio.h>
#include "/include/pcap.h"
#include <math.h>
#include <time.h>
#include <string.h>
#include <stdlib.h>
//#include <include/conio.h>

#define LINE_LEN 16
#define FIN    0x0001
#define SYN    0x0002
#define RST    0x0004
#define PUSH   0x0008
#define ACK    0x0010
#define URG    0x0020
#define ECE    0x0040
#define CWR    0x0080
#define LARGE_NUM 999999999
#define NUM_LOCAL_IPS 3



//FUNCTION PROTOTYPES
void packet_handler(u_char *, const struct pcap_pkthdr *, const u_char *);
void bin_print(int,int);
int chkSYNBit(int);
int chkACKBit(int);
int chkFINBit(int);
void print_src_dst_pairs_tofile();
void print_start_end_tofile();
int print_delays();
int num_active_sessions();              //returns number of active sessions
void initialize_root();
void add_new_session();//struct tcp_session *);     //add new session to LL
void print_all_sessions();              //print LL
void print_completed_sessions();
unsigned int _session_exists(u_short,u_short);//struct ip_address,struct ip_address,
u_short, u_short); //find a session and return its id
int update_status(unsigned int,int);
int check_status(unsigned int);
int set_session_end_time(unsigned int, long double);
int load_local_ip_list();  //load all local ips
int is_local();//struct ip_address,struct ip_address); //chk is current ip is a local ip


//STRUCTURE DEFINITIONS
typedef struct ip_address

{
       u_char byte1;
       u_char byte2;
       u_char byte3;
       u_char byte4;
}ip_address;

/* IPv4 header */
typedef struct ip_header
{
       u_char  ver_ihl;       // Version (4 bits) + Internet header length (4 bits)
       u_char  tos;           // Type of service
       u_short tlen;          // Total length
       u_short identification; // Identification
       u_short flags_fo;      // Flags (3 bits) + Fragment offset (13 bits)
       u_char  ttl;   // Time to live
       u_char  proto;  // Protocol
       u_short crc;   // Header checksum
       struct ip_address      saddr;  // Source address
       struct ip_address      daddr;  // Dest address
```

```
        u_int   op_pad;                // Option + Padding
}ip_header;

/* UDP header*/
typedef struct udp_header
{
        u_short sport; // Source port
        u_short dport; // Destination port
        u_short len;   // Datagram length
        u_short crc;   // Checksum
}udp_header;

/* TCP header*/
typedef struct tcp_header
{
        u_short srcport;      // Source port
        u_short dstport;      // Destination Port
        u_int seq;            // SEQ Number
        u_int ack;            // ACK
        u_short control;      // includes offset, reserved and flags
        u_short window;
        u_short crc;
        u_int opt_pad;
        u_char data[65535];   // Payload
}tcp_header;

/* TCP Session Information Container*/
typedef struct tcp_session {
        unsigned int id;
        ip_address src_add;
        ip_address dst_add;
        u_short src_port;
        u_short dst_port;
        u_char status;
        double start;
        double end;
        struct tcp_session *next;
        struct tcp_session *prev;
}tcp_session;


//GLOBALS
int AVG_PKT_LEN=0;      //
unsigned int TOT_PKT_LEN=0;      // total number of packets
unsigned int NUM_OF_PKTS=0;      //
long double pair_number=0; //index of src-dst pair in array
u_char mask = 0x003F;   //to separate out the bits we need
u_char flags = 0;       //flag bits

//CREATE LL ROOT
struct tcp_session *root; //linked list node
struct ip_header *ih;  //For packet handler
struct tcp_header *th;  //For packet handler
struct tcp_session *new_session; //For add_new_session
struct ip_address _ip_src; //For _session_exists
struct ip_address _ip_dst; //For _session_exists
struct ip_address src; //For is_local
struct ip_address dst;  //For is_local
unsigned int session_id=0;

//END CREATE LL ROOT
int local_ips[NUM_LOCAL_IPS][4]; //for saving local ip list


int main()
{
        //char filename[]= {"CAIDA/equinix-sanjose.dirA.20100325-060200.UTC.anon.pcap"};
//1.66GB OC 192 trace
        //char filename[]= {"CAIDA/equinix-chicago.dirA.20090219-045912.UTC.anon.pcap"};
//72MB OC 48 trace
```

```
        //char filename []={"CAIDA/equinix-sanjose.dirA.20100325-060500.UTC.anon.pcap"};
//700MB OC 192 trace


        //char filename[]= {"CAIDA/equinix-sanjose.dirA.20091217-045904.UTC.anon.pcap"};
//1.66GB OC 192 trace

        //char filename[]= {"CAIDA/equinix-sanjose.dirA.20100325-055905.UTC.anon.pcap"};
//NUST trace
        char filename[]= {"Game Traces/war3-traces/war3-traces/1vs1 Ethereal Trace"};

        pcap_t *fp;
        char errbuf[PCAP_ERRBUF_SIZE];
        char packet_filter[] = "ip and tcp";
        struct bpf_program fcode;
        pcap_if_t *alldevs;

        initialize_root();

        printf("\nRoot initialized...\n");

        /* Open the captured file */
        if((fp = pcap_open_offline(filename, errbuf)) == NULL)
                {
                fprintf(stderr,"\nUnable to open the file %s.\n", filename);
                getchar();
                return -1;
                }

        //compile the filter to only extract TCP packets defined in packet_filter[]
        if (pcap_compile(fp, &fcode, packet_filter, 1, 0) <0 )
                {
                fprintf(stderr,"\nUnable to compile the packet filter. Check the
syntax.\n");
                /* Free the device list */
                pcap_freealldevs(alldevs);
                return -1;
                }
        //set the filter
        if (pcap_setfilter(fp, &fcode)<0)
                {
                fprintf(stderr,"\nError setting the filter.\n");
                /* Free the device list */
                pcap_freealldevs(alldevs);
                return -1;
                }

        //load_local_ip_list();
        printf("\nFilter: %s\n",packet_filter);
        printf("\nReading packets...do not exit...\n");

        pcap_loop(fp, 0, packet_handler, NULL);

        pcap_close(fp);
        printf("\n\nNUM_OF_PKTS=%d, AVG_PKT_LEN=%d",NUM_OF_PKTS,TOT_PKT_LEN/NUM_OF_PKTS);
        //print_src_dst_pairs();
         //print_src_dst_pairs_tofile();
        //print_start_end_tofile();
        //print_delays();
        //print_all_sessions();
        //print_completed_sessions();

        getchar();
        return 0;
}


void packet_handler(u_char *param, const struct pcap_pkthdr *header, const u_char
*pkt_data)
{
        //struct tcp_session *new_session;
        struct tm *ltime;
        char timestr[16];
        //struct ip_header *ih;
        //struct tcp_header *th;
        u_int ip_len;
        u_short sport,dport;
        time_t local_tv_sec;
        int isSYN=0, isACK=0, isFIN=0;


        //New tcp_session node to be added
        new_session = (struct tcp_session *) malloc(sizeof(tcp_session));
         /*
         * unused parameter
         */
        //(VOID)(param);

        /* retireve the position of the ip header */
        ih = (ip_header *) (pkt_data + 14); //length of ethernet header

        /* retireve the position of the tcp header */
        ip_len = (ih->ver_ihl & 0xf) * 4;
        //uh = (udp_header *) ((u_char*)ih + ip_len);
        th = (tcp_header *) ((u_char*)ih + ip_len);

        /* convert from network byte order to host byte order */
        sport = ntohs( th->srcport );
        dport = ntohs( th->dstport );

        /// separate control bits into offset, reserved and flags
        flags = ntohs(th->control);
        flags = flags&mask;
        isSYN=chkSYNBit(flags);
        isACK=chkACKBit(flags);
        isFIN=chkFINBit(flags);
        pair_number = _session_exists(sport,dport);//struct ih->saddr, struct ih-
>daddr,sport,dport);
        /*
        if (pair_number != LARGE_NUM)
                {
                        window_size[pair_number]+=th->window - th->ack;
                        printf("\nW=%ld\tA=%ld\tS=%ld",th->window,th-
>ack,window_size[pair_number]);
                }
        */
         //printf("\n%d--SYN-%d-ACK-%d-FIN-%d\n",pair_number,isSYN,isACK,isFIN);

        if (pair_number == LARGE_NUM)
                {
                //if (isSYN==1){ getchar();}
                if ((isSYN==1) && (isACK==0) && (isFIN==0)) // && is_local())//ih-
>saddr,ih->daddr))
                                {
                                session_id++;

                                //add new session to LL
                                new_session->id=session_id;
                                printf("sanjose.20100325-055905-SRC-DST-DUR-ID:
%u\n",session_id);
                                new_session->src_port=sport;
                                new_session->dst_port=dport;
                                new_session->src_add = ih->saddr;
                                new_session->dst_add = ih->daddr;
                                new_session->status=1;
                                //if (header->ts.tv_usec>1000000) header->ts.tv_sec=header-
>ts.tv_usec % 2;
                                new_session->start = (double)header->ts.tv_usec / 1000000 +
(header->ts.tv_sec);
                                new_session->end=0;
                                //printf("\nSESSION-START: %.8g - \n",new_session->start);
                                add_new_session(new_session);
```

```
                              }
                        }
            else if  ((isSYN==1) && (isACK==1))
                        {
                        update_status(pair_number,2);
                        }
            else if  ((isFIN==1) && (check_status(pair_number) != 5))
                        {

                        set_session_end_time(pair_number,((double)header->ts.tv_usec / 1000000 +
(header->ts.tv_sec)));
                        update_status(pair_number,5);

                        }
            else if ((isSYN==0) && (isFIN==0) && (check_status(pair_number)!=5))
                        {
                        update_status(pair_number,4);
                        }
            pair_number=0;
            TOT_PKT_LEN +=header->len;
            NUM_OF_PKTS++;

}
void bin_print(int num,int bits){

int j;

for(j=bits-1;j>=0;j--)

printf("%i",(num>>j)&01);

}

int chkSYNBit(int num){
        if (num & SYN) return 1;
        return 0;
}

int chkACKBit(int num){
        if (num & ACK) return 1;
        return 0;
}

int chkFINBit(int num){
        if (num & FIN) return 1;
        return 0;
}


void print_src_dst_pairs_tofile()
{
struct tcp_session *temp;
double delay;
char strWrite[100]="";
char temp_str[30]="";
FILE *fp_out;
//fp_out = fopen ("equixnix-sanjose.dirA.20100325-
060200.UTC.anoni.SRC_DST_PAIRS.txt","w");
//fp_out = fopen ("equixnix-chicago.dirA.20090219-
045912.UTC.anoni.SRC_DST_PAIRS.txt","w");
//fp_out = fopen ("eqSan060500_SRC.txt","w");
fp_out = fopen ("OC48-100900.txt","w");



temp=root;
strcat(strWrite,"#PRINT UNIQUE SRC-DST PAIRS\n");
strcat(strWrite,"#SRC_IP      DST_IP  STATUS  DURATION       START   END\n");
while(temp->next!=NULL)
        {
        /////////////////////
        //printf("%d",fp_out==NULL);
        //exit(1);

        //if (ferror(fp_out==NULL))
        //      {
        //      printf("Can't open output file.\n");
        //      exit(1);
        //      }
        //else
        //      {
                if((temp->status==5))// || (temp->status==4))
                        {
                        strcpy(temp_str,"");
                        //itoa(temp->src_add.byte1,temp_str,10);
                        sprintf(temp_str,"%d",temp->src_add.byte1);
                        strcat(strWrite,temp_str);strcat(strWrite,".");
                        //itoa(temp->src_add.byte2,temp_str,10);
                        sprintf(temp_str,"%d",temp->src_add.byte2);
                        strcat(strWrite,temp_str);strcat(strWrite,".");
                        //itoa(temp->src_add.byte3,temp_str,10);
                        sprintf(temp_str,"%d",temp->src_add.byte3);
                        strcat(strWrite,temp_str);strcat(strWrite,".");
                        //itoa(temp->src_add.byte4,temp_str,10);
                        sprintf(temp_str,"%d",temp->src_add.byte4);
                        strcat(strWrite,temp_str);strcat(strWrite,",");
                        //itoa(temp->dst_add.byte1,temp_str,10);
                        sprintf(temp_str,"%d",temp->dst_add.byte1);
                        strcat(strWrite,temp_str);strcat(strWrite,".");
                        //itoa(temp->dst_add.byte2,temp_str,10);
                        sprintf(temp_str,"%d",temp->dst_add.byte2);
                        strcat(strWrite,temp_str);strcat(strWrite,".");
                        //itoa(temp->dst_add.byte3,temp_str,10);
                        sprintf(temp_str,"%d",temp->dst_add.byte3);
                        strcat(strWrite,temp_str);strcat(strWrite,".");
                        //itoa(temp->dst_add.byte4,temp_str,10);
                        sprintf(temp_str,"%d",temp->dst_add.byte4);
                        strcat(strWrite,temp_str);strcat(strWrite,"\t");

                        delay = temp->end - temp->start;
                        //itoa(temp->status,temp_str,10);
                        sprintf(temp_str,"%d",temp->status);
                        strcat(strWrite,temp_str);
                        strcat(strWrite,"\t");
                        sprintf(temp_str, "%.8g", delay);
                        strcat(strWrite,temp_str);strcpy(temp_str,"");

                        strcat(strWrite,"\t");
                        sprintf(temp_str, "%1.10f", temp->start);
                        strcat(strWrite,temp_str);strcpy(temp_str,"");
                        strcat(strWrite,"\t");
                        sprintf(temp_str, "%1.10f", temp->end);
                        strcat(strWrite,temp_str);strcpy(temp_str,"");

                        strcat(strWrite,"\n");

                        printf("\n%s\n",strWrite);

                        fputs(strWrite,fp_out);        //Print the whole record to file
                        strcpy(temp_str,"");
                        strcpy(strWrite,"");
                        }
                        temp=temp->next;
        }
        fclose(fp_out);
}
void print_start_end_tofile()
{
struct tcp_session *temp;
unsigned int i=0;
double delay;
char strWrite[200]="";
char temp_str[30]="";
FILE *fp_out;

//fp_out = fopen ("START_END_CHI_20090115_1-29GB.txt","w");
//fp_out = fopen ("equinix-sanjose.dirA.20100325-060500.UTC.anon.STR_END_DUR.txt","w");
//fp_out = fopen ("equinix-chicago.dirA.20090219-045912.UTC.anon.STR_END_DUR.txt","w");
//fp_out = fopen ("equinix-sanjose.dirA.20100325-060200.UTC.anon.STR_END_DUR.txt","w");

fp_out = fopen ("equinix-sanjose.dirA.20091217-045904.UTC.anon.txt","w");

temp=root;

//sprintf(temp_str,"#NUM_OF_PKTS=%d,
AVG_PKT_LEN=%d",NUM_OF_PKTS,TOT_PKT_LEN/NUM_OF_PKTS);
//strcat(strWrite,temp_str);
strcat(strWrite,"#PRINTING COMPLETED AND RUNNING SESSION START-END PAIRS\n");
strcat(strWrite,"#SERIAL_NUM,START,END,DURATION\n");
while(temp->next!=NULL)
        {
        ////////////////////
        //printf("%d",fp_out==NULL);
        //exit(1);

        //if (ferror(fp_out==NULL))
        //      {
        //      printf("Can't open output file.\n");
        //      exit(1);
        //      }
        //else
        //      {
                if((temp->status==5))// || (temp->status==4))
                        {
                                i+=1;
                        //if (temp->end == 0.0) temp->end=1999999999;
                        sprintf(temp_str,"%u", i);
                        strcat(strWrite,temp_str);                          //serial
number
                        strcat(strWrite,"\t");
                         //printf("\nSTART-%1.10f",temp->start);
                        sprintf(temp_str,"%1.10f", temp->start);
                        strcat(strWrite,temp_str);                          //start time
                        strcat(strWrite,"\t");
                         //printf("END-%1.10f",temp->end);
                        sprintf(temp_str, "%1.10f", temp->end);
                        strcat(strWrite,temp_str);                          //end time
                        strcat(strWrite,"\t");
                        //itoa(temp->status,temp_str,10);                   //status
                        sprintf(temp_str,"%d",temp->status);
                        strcat(strWrite,temp_str);
                        strcat(strWrite,"\t");
                        //if (temp->end == 0.0)
                        //      {
                        //      delay = 0.0;
                        //      }
                        //else
                        //      {
                                delay=temp->end - temp->start;
                        //      }
                        // printf("DURATION-%1.10f\n",delay);
                        sprintf(temp_str, "%1.10f", delay);
                        strcat(strWrite,temp_str);                          //delay


                        strcat(strWrite,"\n");
                        //printf("\n%s",strWrite);
                        //if (connection_status[i] == 5)
                        fputs(strWrite,fp_out);
                        strcpy(temp_str,"");
                        strcpy(strWrite,"");
                        }
                        //}
                        temp=temp->next;
        }
        fclose(fp_out);
}

int print_delays()
{
struct tcp_session *temp;
temp=root;
//temp=temp->next; //skip root (root is empty)
while(temp->next!=NULL)
        {
                //printf("\nPRINTING DELAYS\n%d>%s - %s = %s",temp->id,ctime(temp-
>end),ctime(temp->start),ctime(temp->end-temp->start));
                temp=temp->next;
        }
return 1;
}

int num_active_sessions()
{
struct tcp_session *temp;
int num_active_sessions=0;
temp=root;
//temp=temp->next; //skip root (root is empty)
while(temp->next!=NULL)
        {
                if(temp->status==4) num_active_sessions++;
                temp=temp->next;
        }
return num_active_sessions;
}

void initialize_root()
{
        //root=new tcp_session;
        root = (struct tcp_session *) malloc(sizeof(tcp_session));
        //Initialize root
        root->id=0;
        root->src_add.byte1=0;
        root->src_add.byte2=0;
        root->src_add.byte3=0;
        root->src_add.byte4=0;
        root->src_port=0;
        root->dst_add.byte1=0;
        root->dst_add.byte2=0;
        root->dst_add.byte3=0;
        root->dst_add.byte4=0;
        root->dst_port=0;
        root->start=0;
        root->end=0;
        root->status=0;
        root->next=NULL;
        root->prev=NULL;
        //end initialize root
}
void add_new_session()//struct tcp_session *new_session)
{
struct tcp_session *temp;
temp=root;

while(temp->next != NULL)
        {
                temp=temp->next;
        }
temp->next = new_session;
new_session->prev=temp;
new_session->next=NULL;
}

void print_all_sessions()
```

```
{
struct tcp_session *temp;
double duration=0;
char temp_str[30];
printf("\nLL-UNIQUE SRC-DST PAIRS\n");
temp=root;
while(temp->next!=NULL)
        {
                //duration = difftime(temp->end,temp->start);
                //duration = (clock()-duration)/CLOCKS_PER_SEC;
                //printf("\n%ld-%ld=%ld\n",temp->end,temp->start,temp->end-temp->start);
                duration = temp->end - temp->start;
                sprintf(temp_str, "%.8g", duration);
                printf("%u>%d.%d.%d.%d\t[%d]\t%d.%d.%d.%d\t[%d]\tS=%d\tD=%s\n",temp-
>id,temp->src_add.byte1,temp->src_add.byte2,temp->src_add.byte3,temp->src_add.byte4,temp-
>src_port,temp->dst_add.byte1,temp->dst_add.byte2,temp->dst_add.byte3,temp-
>dst_add.byte4,temp->dst_port,temp->status,temp_str);//,delay);
                duration=0;
                temp=temp->next;
                //_getche();
        }
}
void print_completed_sessions()
{
struct tcp_session *temp;
double duration;
char temp_str[30];
printf("\nLL-UNIQUE SRC-DST PAIRS - COMPLETED\n");
temp=root;
while(temp->next!=NULL)
        {
                if (temp->status == 5)
                        {
                        //duration = difftime(temp->end,temp->start);
                        //duration = (clock()-duration)/CLOCKS_PER_SEC;
                        duration = temp->end - temp->start;
                        sprintf(temp_str, "%.8g", duration);

        printf("%u>%d.%d.%d.%d\t[%d]\t%d.%d.%d.%d\t[%d]\tS=%d\tD=%s\n",temp->id,temp-
>src_add.byte1,temp->src_add.byte2,temp->src_add.byte3,temp->src_add.byte4,temp-
>src_port,temp->dst_add.byte1,temp->dst_add.byte2,temp->dst_add.byte3,temp-
>dst_add.byte4,temp->dst_port,temp->status,temp_str);
                        duration=0;
                        //_getche();
                        }
                temp=temp->next;
        }
}
unsigned int _session_exists(u_short sport, u_short dport)//struct ip_address _ip_src,
struct ip_address _ip_dst, u_short sport, u_short dport)
{
struct tcp_session *temp;
temp=root;
//temp=temp->next; //skip root (root is empty)
while(temp->next!=NULL)
        {
                 if ((temp->src_add.byte1 == ih->saddr.byte1) && (temp->src_add.byte2 ==
ih->saddr.byte2) && (temp->src_add.byte3 == ih->saddr.byte3) && (temp->src_add.byte4 ==
ih->saddr.byte4))

                        {
                        if ((temp->dst_add.byte1 == ih->daddr.byte1) && (temp-
>dst_add.byte2 == ih->daddr.byte2) && (temp->dst_add.byte3 == ih->daddr.byte3) && (temp-
>dst_add.byte4 == ih->daddr.byte4))
                                {
                                if ((temp->src_port == sport) && (temp->dst_port == dport))
                                        {
                                        return temp->id;
                                        }
                                }
                        }
```

```
               else if ((temp->src_add.byte1 == ih->daddr.byte1) && (temp->src_add.byte2
== ih->daddr.byte2) && (temp->src_add.byte3 == ih->daddr.byte3) && (temp->src_add.byte4
== ih->daddr.byte4))
                       {
                       if ((temp->dst_add.byte1 ==  ih->saddr.byte1) && (temp-
>dst_add.byte2 ==  ih->saddr.byte2) && (temp->dst_add.byte3 ==  ih->saddr.byte3) &&
(temp->dst_add.byte4 ==  ih->saddr.byte4))
                               {
                               if ((temp->src_port == dport) && (temp->dst_port == sport))
                                       {
                                       return temp->id;
                                       }
                               }
                       }
               temp=temp->next;
       }
return LARGE_NUM;
}

int update_status(unsigned int session_number,int new_status)
{
struct tcp_session *temp;
temp=root;
//temp=temp->next; //skip root (root is empty)
while(temp->next!=NULL)
       {
               if (temp->id == session_number)
                       {
                       temp->status=new_status;

                       if (new_status == 5)
                               {
                               //////////////////////WRITE FINISHED SESSION RECORD TO
FILE
                               char strWrite[100]="";
                               char temp_str[30]="";
                               double delay;
                               FILE *fp_out;
                               fp_out = fopen ("data003.txt","a+");

                               strcpy(temp_str,"");
                               sprintf(temp_str,"%d",temp->src_add.byte1);
                               strcat(strWrite,temp_str);strcat(strWrite,".");
                               sprintf(temp_str,"%d",temp->src_add.byte2);
                               strcat(strWrite,temp_str);strcat(strWrite,".");
                               sprintf(temp_str,"%d",temp->src_add.byte3);
                               strcat(strWrite,temp_str);strcat(strWrite,".");
                               sprintf(temp_str,"%d",temp->src_add.byte4);
                               strcat(strWrite,temp_str);strcat(strWrite,"\t");
                               sprintf(temp_str,"%d",temp->dst_add.byte1);
                               strcat(strWrite,temp_str);strcat(strWrite,".");
                               sprintf(temp_str,"%d",temp->dst_add.byte2);
                               strcat(strWrite,temp_str);strcat(strWrite,".");
                               sprintf(temp_str,"%d",temp->dst_add.byte3);
                               strcat(strWrite,temp_str);strcat(strWrite,".");
                               sprintf(temp_str,"%d",temp->dst_add.byte4);
                               strcat(strWrite,temp_str);strcat(strWrite,"\t");

                               delay = temp->end - temp->start;
                               sprintf(temp_str,"%d",temp->status);
                               strcat(strWrite,temp_str);
                               strcat(strWrite,"\t");
                               sprintf(temp_str, "%.8g", delay);
                               strcat(strWrite,temp_str);strcpy(temp_str,"");

                               strcat(strWrite,"\t");
                               sprintf(temp_str, "%1.10f", temp->start);
                               strcat(strWrite,temp_str);strcpy(temp_str,"");
                               strcat(strWrite,"\t");
                               sprintf(temp_str, "%1.10f", temp->end);
                               strcat(strWrite,temp_str);strcpy(temp_str,"");
```

```
                              strcat(strWrite,"\n");

                              //printf("\n%s\n",strWrite);

                              fputs(strWrite,fp_out);       //Print the whole record to
file
                              strcpy(temp_str,"");
                              strcpy(strWrite,"");

                              fclose(fp_out);
                              /////////////////////////END WRITE RECORD TO FILE
                              }
                      return 1;
                      }
              temp=temp->next;
       }
return 0;
}

int check_status(unsigned int session_number)
{
struct tcp_session *temp;
temp=root;
//temp=temp->next; //skip root (root is empty)
while(temp->next!=NULL)
       {
              if (temp->id == session_number)
                     {
                     return temp->status;
                     }
              temp=temp->next;
       }
return 0;

}
int set_session_end_time(unsigned int session_number, long double end_time)
{
struct tcp_session *temp;
temp=root;
//temp=temp->next; //skip root (root is empty)
while(temp->next!=NULL)
       {
              if (temp->id == session_number)
                     {
                     temp->end=end_time;//time(NULL);
                     return 1;
                     }
              temp=temp->next;
       }
return 0;
}

int load_local_ip_list()
{
       local_ips[0][0]=0;
       local_ips[0][1]=0;
       local_ips[0][2]=112;
       local_ips[0][3]=2;
       local_ips[1][0]=0;
       local_ips[1][1]=0;
       local_ips[1][2]=96;
       local_ips[1][3]=2;
       local_ips[2][0]=0;
       local_ips[2][1]=0;
       local_ips[2][2]=128;
       local_ips[2][3]=2;
return 1;
}
int is_local()//ip_address src, ip_address dst)
{
        int i;
        for (i=0;i<NUM_LOCAL_IPS;i++)
                {
                        if((src.byte1 == local_ips[i][0]) && (src.byte2 == local_ips[i][1])
&& (src.byte3 == local_ips[i][2]) && (src.byte4 == local_ips[i][3]))
                                return 1;
                        else if((dst.byte1 == local_ips[i][0]) && (dst.byte2 ==
local_ips[i][1]) && (dst.byte3 == local_ips[i][2]) && (dst.byte4 == local_ips[i][3]))
                                return 1;
                        }
//src.byte1=NULL;src.byte2=NULL;src.byte3=NULL;src.byte4=NULL;
//dst.byte1=NULL;dst.byte2=NULL;dst.byte3=NULL;dst.byte4=NULL;

return 0;
}
```