A COST-EFFECTIVE DISTRIBUTED ARCHITECTURE FOR CONTENT DELIVERY AND EXCHANGE OVER EMERGING WIRELESS TECHNOLOGIES

by

Khondkar R. Islam

A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
in Partial Fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Information Technology

Committee:

______________________________	Dr. J. Mark Pullen, Dissertation Director
______________________________	Dr. Jeremy E. Allnutt, Committee Member
______________________________	Dr. Edgar H. Sibley, Committee Member
______________________________	Dr. Charles M. Snow, Committee Member
______________________________	Dr. Stephen Nash, Senior Associate Dean
______________________________	Dr. Kenneth S. Ball, Dean, Volgenau School of Engineering
Date: __________________________	Spring Semester 2013 George Mason University Fairfax, VA

A Cost-Effective Distributed Architecture for Content Delivery and Exchange Over Emerging Wireless Technologies

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

by

Khondkar R. Islam
Master of Science
American University, 1991

Director: J. Mark Pullen, Professor
Department of Computer Science

Spring Semester 2013
George Mason University
Fairfax, VA

339

DEDICATION

This work is dedicated to my loving wife Joya and my parents.

ACKNOWLEDGEMENTS

I would like to express my gratitude to my dissertation director, Dr. Mark Pullen for his continuous and sincere support with this research whose guidance from the beginning to end has been invaluable. I am grateful to the Chair of Department of Applied IT, Dr. Donald Gantz, where I am also employed for his motivation toward my work and flexibility that enabled me to focus on this arduous task. Dr. Charles Snow, Associate Chair for Graduate Studies of the Department of Applied IT and a member of my dissertation committee has been by my side from the beginning in providing direction for the completion of this work. Committee members, Dr. Jeremy Allnutt and Dr. Edgar Sibley, have also been supportive toward the progress of my research work. My wife Joya has been very patient with my hard work that required late night and weekend hours of effort over a period of several years. Her encouragement helped me sustain the dedication and momentum to complete my doctoral degree. The affection and support of my parents motivated me and kept me going through frustrating times in earning this prestigious terminal degree. To the many others who provided help and support that I have not named here, I extend my thanks.

TABLE OF CONTENTS

Page

List of Tables… x

List of Figures xi

List of Abbreviations xiii

Abstract………. xv

CHAPTER 1: INTRODUCTION 1

1.1 Background 1

1.2 Barriers to DE 4

1.3 Motivation 10

1.4 Options Available for the Target Architecture 11

1.5 Contributions and Summary 13

1.5.1 Contributions 13

1.5.2 Summary 15

CHAPTER 2: LITERATURE REVIEW OF P2P SYSTEMS AND ARCHITECTURES 17

2.1 Introduction 17

2.2 P2P Systems and Architectures 18

2.3 Unstructured Systems 19

2.3.1 Hybrid Decentralized 19

2.3.2 Purely Decentralized 20

2.3.3 Partially Centralized 23

2.3.4 BitTorrent 24

2.4 Structured Systems 26

2.4.1 Freenet- Hybrid Architecture 27

2.4.2 Chord Architecture 30

2.4.3 CAN Architecture 34

2.4.4 Tapestry Architecture 35

2.4.5 Structured Architecture Problems 37

2.5 Two-tier Overlay Multicast Architecture (TOMA)... 37
2.6 Reliable Overlay Multicast Architecture (ROMA)... 40
2.7 Overcast Architecture.. 41
2.8 Narada Brokering Architecture 42
2.9 Distance Education System Architecture 45
2.10 Summary of Architectures.. 48
2.11 Conclusion... 50
CHAPTER 3: P2P AND OVERLAY MULTICAST TECHNOLOGIES 52
3.1 Introduction 52
3.2 Nemo 57
3.3 FatNemo 59
3.4 Mithos and HyperCast.. 61
3.5 Pastry 62
3.6 Learned Model 63
3.7 SplitStream, Magellan and Magellan-Nemo 64
3.8 ENarada 67
3.9 Parameter-Free Peer Selection Algorithm.. 69
3.10 Chunkyspread Algorithm 71
3.11 Conclusion.. 74
CHAPTER 4: LEARNING MANAGAMENT SYSTEMS AND SYNCHRONOUS DISTANCE EDUCATION TOOLS 80
4.1 Introduction 80
4.2 DE LMS and SDET Requirements 80
4.3 Communication Tools 82
4.4 Grade book Component... 82
4.5 Knowledge Assessment... 83
4.6 Administrator Role 83
4.7 WebCT, Intralearn, AulaNet, UniverSite and Col DE tools 84
4.8 Blackboard and Moodle 88
4.9 Synchronous DE: Network EducationWare (NEW) 92
4.10 Moodle Integrated Synchronous Teaching/Conferencing (MIST/C)................ 95
4.11 Elluminate vs. Macromedia Breeze.. 98
4.12 Elluminate vs. Dimdim.. 100

4.13 BigBlueButton 104
4.14 Conclusion 105
CHAPTER 5: THE CONTENT DELIVERY PROBLEM 106
5.1 Introduction 106
5.2 Client-Server Model 106
5.2.1 The Client-Server Problem 109
5.2.2 Tests using Client-Server Model 112
5.3 The IP Multicast Problem 114
5.3.1 Routing Protocols 116
5.4 The Communications Media Problem 117
5.5 Conclusion 119
CHAPTER 6: REQUIREMENTS 120
6.1 Introduction 120
6.2 Low Cost 120
6.3 Support any Language 121
6.4 Limited Capacity Network 122
6.5 Single Point of Failure 122
6.6 Distributed Nodes 122
6.7 Technologies that Meet the Requirements 123
6.7.1 P2P Distributed Nodes 124
6.7.2 Overlay Multicast 125
6.7.3 P2P Distributed Nodes with Overlay Multicast Functionality 127
6.8 Conclusion 128
CHAPTER 7: ARCHITECTURE FRAMEWORK 129
7.1 Introduction 129
7.2 Content Delivery Framework 131
7.3 Group Formation 132
7.4 Content Distribution 133
7.5 Search Process 134
7.6 Presentation Methodology 136
7.7 Content Delivery Architecture 136
7.7.1 Access Layer 138

7.7.2 Presentation Layer 138
7.7.3 Build and Search Layer 139
7.7.4 Data Layer 142
7.8 Conclusion 143
CHAPTER 8: EXPERIMENTS TOWARD THE DESIGN OF THE SELECTED ARCHITECTURE 144
8.1 Introduction 144
8.2 Parameters 146
8.2.1 Link Stress 147
8.2.2 End-to-End Delay 147
8.2.3 Transient Nodes 148
8.2.4 Query Resolution 149
8.2.5 Content Replication 149
8.2.6 Single Point of Failure 149
8.2.7 Load Balancing 150
8.3 Data Bit Rate 151
8.4 Top-Down Ordered Structure 154
8.5 Scalability of P2P System 156
8.6 File and Fragment Size Evaluation 159
8.7 Physical Link Data Rate Evaluation 161
8.8 Performance Optimization by Peer Size Percentage 164
8.9 Evaluation of Larger Network, Peerset and File Sizes with Small Percentage of Peers 167
8.10 Impact of Seeders on Performance 170
8.11 Peer Dynamics 172
8.12 Impact of Duplicate Requests 172
8.13 Swarm Size Evaluation 173
8.14 *n*-Tier Hierarchical Training Model 174
8.15 Conclusion 180
CHAPTER 9: CONCLUSIONS, CONTRIBUTIONS AND FUTURE DIRECTIONS 182
9.1 Conclusions 182
9.2 Contributions 184
9.3 Future Directions 186

REFERENCES…… 189
APPENDIX A: CONGESTION WINDOW SIMULATION CODE 200
APPENDIX B: P2P SIMULATION CODE.. 204

LIST OF TABLES

Table Page
Table 1: Percentage of Internet users 2
Table 2: Synchronous DE delivery challenges 7
Table 3: Characteristics of synchronous and asynchronous DE delivery 8
Table 4: Finger table of node 6 of Chord ring in Figure 6 32
Table 5: Summary of P2P systems and overlay multicast protocols 76
Table 6: Summary of five distance education tools 86
Table 7: Comparison of Bb and Moodle LMS 92
Table 8: MIST/C hardware and software requirements 96
Table 9: MIST/C functional requirements 96
Table 10: Breeze and Elluminate features and functionalities 98
Table 11: Advantages and disadvantages of Breeze and Elluminate 99
Table 12: Features of Elluminate 9.0 and Dimdim 4.5 101
Table 13: Participant feedback on Elluminate and Dimdim 102
Table 14: Advantages and disadvantages of Elluminate and Dimdim 103
Table 15: Simulation results of 90, 95 and 99% peers, 200 kb/s links, 100 MB file 158
Table 16: Simulation results of 1, 5 and 10% peers, 50 kb/s links, 50 MB file 160
Table 17: Simulation results of 1, 5 and 10% peers, 100 kb/s links, 50 MB file 162
Table 18: Simulation results of 80 and 90% peers, 50 kb/s links, 50 MB file 165
Table 19: Optimum performance with 100 nodes 166
Table 20: Simulation results of 15 and 20% peers, 100 kb/s links, 100 MB file 168
Table 21: Optimum performance with 500 nodes 169
Table 22: Simulation results of 99, 95, 90, 80 and 70% peers and 1% seeders, 200 kb/s links, 100 MB file 171
Table 23: Simulation results of 99, 95, 90, 80 and 70% peers and 2% seeders, 200 kb/s links, 100 MB file 171
Table 24: Simulation results of 99, 95, 90, 80 and 70% peers and 5% seeders, 200 kb/s links, 100 MB file 171
Table 25: Simulated results of peer dynamics 172
Table 26: Simulation results of duplicate request packets 173
Table 27: Simulation results of swarm sizes 174

LIST OF FIGURES

Figure Page

Figure 1: Distribution of out-of-school children by region, 1999 and 2009 (Percentage) . 3

Figure 2: Hybrid decentralized architecture 20

Figure 3: Purely decentralized architecture (Gnutella) search process - step 1 22

Figure 4: Purely decentralized architecture (Gnutella) search process- step 2 22

Figure 5: Loosely structured architecture (Freenet) search process 29

Figure 6: Structured architecture - Chord system 31

Figure 7: Chord key lookup algorithm..... 33

Figure 8: MASS architecture for e-assessment..... 46

Figure 9: Three-tier client-server architecture 108

Figure 10: Load balanced web servers..... 109

Figure 11: Notional inverse relationship between number of users and data bit rate with the client-server model 111

Figure 12: Notional relationship between latency and link capacity with the client-server model 111

Figure 13: TCP congestion window with 32 KB file 113

Figure 14: TCP congestion window with 128 KB file 114

Figure 15: IP multicast problem 116

Figure 16: Group formation 132

Figure 17: Content distribution..... 134

Figure 18: Search process 135

Figure 19: P2P Overlay Content Delivery (POCD) architecture..... 138

Figure 20: POCD architecture search example..... 140

Figure 21: Notional linear relationship between user population and file availability with P2P systems..... 151

Figure 22: Notional non-linear relationship between number of users and data bit rate with P2P systems 152

Figure 23: Notional relationship between user population and average latency with traditional P2P systems 153

Figure 24: Top-Down ordered structure of simulation experiments 154

Figure 25: Comparison with different fragment sizes and network link capacity..... 159

Figure 26: Time to share 50 MB file using 50 kb/s links in 100 node 161

Figure 27: Time to share 50 MB file using 100 kb/s links in 100 node 162

Figure 28: Download and upload values of Node 1 163

Figure 29: Time to share 50 MB file using 50 kb/s links in 100 node 165

Figure 30: Best-of-the-best results of 100 nodes 166

Figure 31: Time to share 100 MB file using 100 kb/s links in 500 node 168
Figure 32: Best-of-the-best results of 500 nodes 170
Figure 33: Growth of Internet use: wired vs. wireless worldwide................................ 176
Figure 34: *n*-tier hierarchical training model 177
Figure 35: Notional data bit rate map 180

LIST OF ABBREVIATIONS

2-Hop ORA- 2-Hop Overlay Routing Algorithm
3G- 3rd Generation
4G- 4th Generation
ACK- Acknowledgement
Bb- Blackboard
BBB- BigBlueButton
BDN- Broker Discovery Nodes
BGMP- Border Gateway Multicast Protocol
BT- BitTorrent
CA- Coordination Agent
CAN- Content Addressable Network
CBT- Core Based Tree
CN- Corresponding Node
Col (Online Courses)
CPU- Central Processing Unit
CS- Central Server
CSS- Central Storage Server
DE- Distance Education
DP- Downloading Peer
DVMRP- Distance Vector Multicast Routing Protocol
E2E- End-to-end
EDGE- Enhanced Data Rates for GSM Evolution
FEC- Forward Error Correction
FF- File Fragment
GT-ITM- Georgia Tech Internetwork Topology Model
GUI- Graphical User Interface
ICPQRL- Independent Commission for Population and Quality of Life
IP- Internet Protocol
ISP- Internet Service Provider
IT- Information Technology
ITU- International Telecommunications Union
JN- Joining Node
LLT- Lower Latency Threshold
LMS- Learning Management System
MASS- Modular Assessment System for Modern Learning Settings
MDG- Millennium Development Goals
MinL- Minimum Load
MIST/C- Moodle Integrated Synchronous Teaching and Conferencing
MND- Minimum Node Degree

MoM- Message Oriented Middleware
MOSPF- Multicast Open Shortest Path First
MSDP- Multicast Source Discovery Protocol
MSON- Multicast Service Overlay Network
N- Node
NACK- Negative Acknowledgement
NAT- Network Address Translation
NB- Narada Brokering
NEW- Network Education Ware
NN- Neighboring Node
NS- Nodes Storing
OLAMP- Overlay Aggregated Multicast Protocol
OMRS- Overlay Multicast Relay Server
OMS- Overlay Multicast Server
OMSE- Overlay Multicast Server Environment
ON- Ordinary Node
OSI- Open Systems Interconnection
OSPF- Open Shortest Path First
P2P- Peer-to-Peer
PC- Personal Computer
PIM-SM- Protocol Independent Multicast Sparse Mode
POCD- P2P Overlay Content Delivery
PSTN- Public Switched Telephone Network
QoS- Quality of Service
RN- Requesting Node
ROMA- Reliable Overlay Multicast Architecture
RP- Rendezvous Point
RPC- Remote Procedure Call
RTT- Round Trip Time
SDET- Synchronous Distance Education Tool
SHA-1- Secure Hash Algorithm-1
SIS- Student Information System
SN- Super node
SOA- Service Oriented Architecture
SOAP- Simple Object Access Protocol
SSL- Secure Sockets Layer
TCP- Transmission Control Protocol
TLM- Transport Layer Multicast
TOMA- Two-tier Multicast Architecture
TS- Tracker Server
TTL- Time-to-Live
UDP- User Datagram Protocol
ULT- Upper Latency Threshold
UN- United Nations
UP- Uploading Peer
WiMAX- Worldwide Interoperability for Multiple Access
WS- Web Server
XML- Extensible Markup Language

ABSTRACT

A COST-EFFECTIVE DISTRIBUTED ARCHITECTURE FOR CONTENT DELIVERY AND EXCHANGE OVER EMERGING WIRELESS TECHNOLOGIES

Khondkar R. Islam, Ph.D.

George Mason University, 2013

Dissertation Director: Dr. J. Mark Pullen

Opportunities in education are lacking in many parts of the developed nations and are missing in most parts of the developing nations. This is, in significant part, due to shortages of classroom instructional resources such as quality teaching staff, hardware and software. Distance education (DE) has proved to be a successful teaching approach and overcomes some of the barriers imposed by classroom instruction, primarily due to the shortage of teachers.

Many DE software tools have been developed and are in use today. Most require high network capacity, not supported by common long distance wireless network infrastructures in many places of the world. To address obstacles related to network infrastructures of developing countries for content delivery and exchange, this research develops the design of a cost-effective distributed architecture for content delivery and exchange over emerging limited capacity wireless technologies. The design of the proposed target architecture includes an overlay network with distributed peer-to-peer systems. Simulation is used to explore parameters and metrics in order to validate the effectiveness and scalability of this architecture. An *n*-tier hierarchical training model to train local instructors by experienced instructors, using DE resources supported by this architecture, is discussed as a way to mitigate the teacher shortages in developing countries. The situation in Bangladesh is used to provide examples, based on the author's familiarity with it.

CHAPTER 1: INTRODUCTION

1.1 Background

Distance education (DE) is becoming more popular not only in the developed nations, but also in developing countries. There have always been those who question whether DE is an effective delivery method. That is, do students at a remote location learn as much as students in a traditional classroom [1]? Research in this area has validated that student learning outcomes via DE delivery methods are equivalent to students learning in a traditional classroom [2].

In developing nations today there is a shortage of quality teaching staff. This has been one of the main factors for the slow growth of primary education in those countries. DE can help to address this problem.

According to [3], as of 31 March 2011, the world population stood at 6.930 billion and total number of Internet users was 2.095 billion, 30.2% of the entire population. As of 31 March 2011, the United States total population was about 310 million and total number of Internet users was 239 million, 77.3% of the total population [4]. This is a significantly higher percentage of Internet users in the United States in comparison to the world overall. The discrepancy is even greater as we look at the data of Bangladesh, a developing nation in Southeast Asia. It has a total population of about 158 million and total number of Internet users is about 617 thousand, which is 0.4% of the population [5]. Table 1 summarizes this data.

Table 1: Percentage of Internet users [3]–[5]

Location	Population	Internet Users	% of Population
World	6,930,055,154	2,095,006,005	30.2
United States	310,232,863	239,232,863	77.3
Bangladesh	158,065,841	617,300	0.4

In the United States, during 2003-2009, the percentage of online higher education enrollments grew faster than the total percentage of higher education student population. During the Fall 2008 semester, more than 4.6 million students took at least one online course at degree granting institutions, which was a 17 percent jump from Fall 2007 semester. There was only 1.2 percent growth of overall higher education student population during these semesters. The survey also found more than 25 percent of higher education students took at least one online course [6]. There were 4,160 two-year and four-year Title IV institutions during the years 2006-2007 and approximately 66 percent of those institutions offered at least one DE course. There were an estimated 11,200 DE programs and about 12.2 million DE student enrollments. Of these 12.2 million students, 77 percent were online, 12 percent were hybrid and 10 percent were via some traditional DE mode [7].

In Bangladesh, government and international education agencies are striving to impart primary, secondary, and Information Technology (IT) education to the underserved. This is also consistent with the second of eight goals of the United Nations (UN) Millennium Development Goals (MDG) [8], [9]. There are underserved students between the ages of 10 and 20 who live in remote areas such as villages and small townships where primary education is very poor as the student to teacher ratio is large (50:1) and there is a severe lack of competent teaching staff. It is worse in sub-Saharan African countries where the ratio is 80:1 [10]. If this goal is achieved, the literacy rate of the country would improve, gradually alleviating poverty, and be the key driver for economic growth [11], [12]. It is worth noting that access to primary education has improved from the time MDG 2 were established because primary school enrollment rose worldwide from 82% in 1990 to 89% in 2008 [13]. Despite this rise, there are about 67 million children not in school, of which 32 million are in sub-Saharan Africa [9], [14]. Shortage of one million instructors, in sub-Saharan Africa alone is a huge obstacle in enrolling a percentage of the 32 million students [10]. Figure 1 shows that the majority of the underserved are from developing countries.

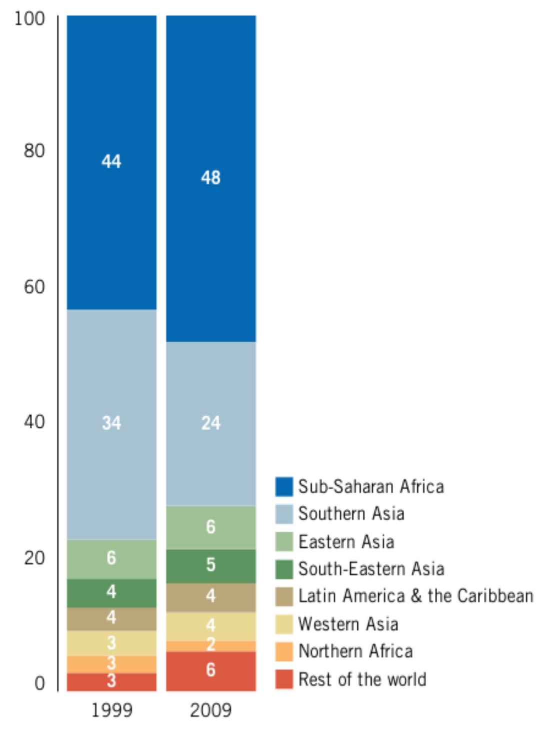

Figure 1: Distribution of out-of-school children by region, 1999 and 2009 (Percentage) [14]

Paprock in [15] points out that past research demonstrates the importance of basic education for better health and quality of life, productivity, higher well-being of women and improved government. He also cites Independent Commission for Population and Quality of Life (ICPQL), which state “education is one of the keys to social development, and virtually every aspect of the quality of life” [15].

DE teaching methods may be used to address this problem. For DE to be successful, the participants must have access to a shared network. Plus, the network connectivity has to offer adequate data bit rate and have high availability standards. For synchronous DE, the bit rate has to be at least 56 kb/s [16], since distance education tools available today require at a minimum this much data capacity for transmitting audio and graphics. With video, the requirement is more [16]–[19] such as broadband Internet connection that involve data rates at or above 800 kb/s [20]. My research will address how such levels of network capacity can be provided in developing nations today in an affordable way.

1.2 Barriers to DE

There are three important barriers to DE pointed out by Paprock in [15]:

1. Lack of resources needed for meaningful development and sustenance of technology based learning;
2. Lack of infrastructure (which includes information and communication hardware systems) to support modern technologies in least developed and/or low technology countries;
3. Lack of recurrent funding necessary to acquire or develop appropriate software and courseware on a continuous basis, and maintain service and replace the equipment.

Wireless communication has come a long way in developing countries. They were able to leapfrog the developed countries by taking advantage of this technology to improve their data communications infrastructure and meet the growing needs of their vast population. For example, in Bangladesh there are more than 70 million users of cellular phones [21] and 617,300 Internet users [5]. Ten years earlier, only the most affluent users could afford telephones and it was a dream for the general population to have access to such equipment and technology. My research looks forward to an architecture entailing similar growth with regard to Internet access that can be used for DE, again by leapfrogging technology.

Worldwide Interoperability for Microwave Access (WiMAX) is the IEEE 802.16 long-range networking standard for mobile broadband Internet service. This technology is suitable for offering Internet service in rural areas. The service was recently introduced in Bangladesh but is facing technical difficulties, because its signal is interfering with several television channel stations since it operates in the 3.5 GHz band [22]. Sprint introduced its first WiMAX cell phone at the Cellular Telecommunications Industry Association (CTIA) 2010 event [23]. Since the approval of Long Term Evolution (LTE) standard in December 2008, Sprint has moved away from WiMAX to LTE. This is because LTE is more compatible with existing cellular devices and offers up to 100 Mb/s download rate. WiMAX download data bit rate is up to 40 Mb/s to fixed broadband applications [24]. However, many developing countries cannot think of 4G LTE because they do not have 3G cellular service in place yet.

INMARSAT's Broadband Global Area Network (BGAN) operates at data rates up to 492 kb/s. It uses a new generation of I-4 advanced communications satellites. Users with a lightweight satellite terminal can access the Internet from almost anywhere in the world [25]. It is cost-effective for some applications where other Internet service options are not available, because the broadband satellite user terminal costs between $1800 to $5000 and service cost is about $5.00 per MB [26]. This satellite service is suitable for companies or individuals in locations where terrestrial Internet service is not available, but is not a feasible option for DE content delivery and exchange because it is cost prohibitive for this application, where rates of $7.50/GB are an economic necessity [27].

Internet is the network of choice for DE, since local instructors and students at remote locations would access education material stored at distance locations over the Internet using their distance education client application software on their local computer. They may also use mobile devices to access education material over the Internet via their cellular network service provider. Smartphones are about 30% of all mobile phones sold today, which is 11% more than the number of units sold in 2008 [14]. It is estimated that the shipments will more than double from 494 million units in 2011 to over one billion units in 2015 [14], [28]. There is a caveat to this: the mobile devices are cost-prohibitive for the local students. The alternative is to train local instructors using DE delivery who would then be ready to teach students in live classrooms at their location in classroom settings. This model would also be applicable for students who can afford mobile devices because they would be able to access lecture sessions and materials directly.

There are two general types of DE delivery methods: 1) *Asynchronous* and 2) *Synchronous*. With asynchronous delivery, instructors can record lectures that are stored in a server or prepare lessons as web pages. Students access the server at their individual convenience to retrieve the lectures. Home assignments, exams and other class materials are also uploaded to the server. Synchronous distance education is achieved by distributing in-class sessions using telecommunications so that students attend online classes together during the class time. They participate in lectures, view slide presentations and interact with the instructor and other students via the Internet using the distance education client application software installed on their computer. This creates an environment where the students gain most benefits of attending a live classroom without having to actually go to a classroom. It is worth noting that video streaming generally is not mandatory, since in many cases, audiographics can convey the information that is needed. Synchronous video for DE delivery has several tradeoffs and challenges [1]. It requires high capacity network services for reliable video streaming [17]. Plus, audio and video are sometimes not synchronized which leads to confusion since lip movement and audio being heard is not the same. Further, low video resolution to conserve network capacity and small display screens of Learning Management Systems (LMS) and Synchronous Distance Education Tools (SDET) do not show clear view of facial expressions, without which there is no point providing the video [29]. Table 2 is a summary of synchronous DE challenges.

Table 2: Synchronous DE delivery challenges

Synchronous DE Challenges	Requires high network capacity Internet service (at least 200 kb/s)	Audio and video are not synchronized during network congestion [29]	Audio and video transmission disrupted due to server failure and/or network congestion	Forced to stream low video resolution to conserve network capacity [29]	Technological interruptions distracted students and instructors, which led to lack of student participation [29]

Research shows students do not get the opportunity to interact with the instructor and other students while they are in an asynchronous learning environment. Some students are confused about the assignments and course objectives, and feel frustrated and isolated. On the other hand, despite the challenges associated with synchronous education, it approximates face-to-face dialog and promotes a sense of community. Smith [1] makes a case that overall student outcome is also better with synchronous education over asynchronous learning. This is because students are motivated since synchronous education makes the courses more engaging.

However, it is important to note, Hrastinski's research results in [30] show, as described above, there are benefits and limitations associated with both asynchronous and synchronous DE delivery methods, and the two complement each other. This is because with the webpage form of asynchronous DE delivery, there is an increase in cognitive participation to deal with complex issues because the remote student has ample time to analyze and understand the message that does not require immediate response. On the contrary, with synchronous DE method, there is a greater level of personal participation that increases psychological arousal because the remote student is motivated to read the messages, and participate with the instructor and other remote students with less complex subject matters since immediate response is expected. Both DE delivery modes are included in this work because local instructors and students in Bangladesh or any country may take advantage of both asynchronous and synchronous DE delivery and exchange methods to learn from scarce quality teachers. Characteristics of synchronous and asynchronous DE delivery are presented in Table 3.

Table 3: Characteristics of synchronous and asynchronous DE delivery [30]

Characteristics	**Synchronous DE**	**Asynchronous DE**
Positive	Increases psychological arousal	Increases cognitive participation
Negative	Does not increase cognitive participation	Personal participation is low

Quality teachers do not necessarily have to be local. They can be teaching from any part of the world. There may be a good supply of quality teachers in an urban area, but only a few in remote rural areas [31]. Using a combination of synchronous and asynchronous distance education delivery methods, the experienced teachers can teach the students not only in their local area but also in the remote rural areas. As mentioned earlier, this method also can be used to train the trainers where quality teachers can train inexperienced teachers in remote areas to better prepare them to teach local students in classrooms, which would benefit students without access to distance education tools to learn via DE delivery, because they do not have a computer and/or Internet service. Both DE delivery methods may also be used to train remote teachers on using distance education tools, and enable them to use the DE delivery methods for teaching students in remote areas. Hentea et al. [17] indicate it is important to train the instructors to use computers, emerging networking technologies and distance education tools. This would help achieve rapid expansion of DE delivery because growth of this mode of education has proven to be enormous.

In the *hybrid* learning model, web-based DE is blended with face-to-face teaching sessions to ensure online education equates to the quality of on-campus programs. Research studies show web-based DE students perform better than classroom students with exams, however, classroom student performance was better with home assignments and projects [17]. Thus, hybrid DE appears to offer greater effectiveness as it includes periodic face-to-face interaction between the instructor and students. Quality instructors can teach remote local students via web-based synchronous and asynchronous distance education delivery modes, and also train remote local instructors to help them improve their teaching quality, so they are able to teach the local students in classrooms at the remote location.

1.3 Motivation

With the available wireless Internet service in Bangladesh and most developing nations, synchronous DE delivery is not practical using traditional network architecture using a client-server model. Traditional network architecture refers to the reliance on the Internet Protocol (IP) layer for electronic packet addressing and routing. Both synchronous and asynchronous DE today are based on the client-server architecture, where learning material needs to be stored in a server and live lectures have to be streamed to the students via that central server. It taxes the data network and server to transmit and process large volume of data. Synchronous DE delivery is impractical in low capacity network environments, as it requires at least 56 kb/s data rate Internet service over cable and wireless networks. DE delivery becomes unaffordable for developing nations in a client-server environment because it needs clusters of high-end servers to meet the processing and storage needs of DE delivery. It is cost-prohibitive for developing countries to provision wireline broadband Internet infrastructure for efficient data packet forwarding and routing via expensive cable lines and routing equipment. The other option is wireless broadband. This is being provisioned in developing countries but progress has been slow due to economical, political and regulatory obstacles [20].

Due to the influx of mobile devices, there is an opportunity to deliver DE content via limited capacity wireless network service to end clients with smart phones and tablet computers. To meet the projected increase in number of end users, the target architecture must be scalable. This is not possible with the traditional network architecture in place today because, as indicated earlier, broadband wireless Internet service is not available.

1.4 Options Available for the Target Architecture

Based on my research and requirements established in chapter 6, several options may be considered to address problems just discussed. Feasibility of each option is presented after the list.

1. Revamp the Public Switched Telephone Network (PSTN) infrastructure to offer broadband Internet service to remote locations.
2. Offer nationwide 3rd generation (3G) or 4th generation (4G) cellular service in developing countries to achieve high speed Internet service.
3. Ensure that routers in the Internet are multicast enabled to forward IP multicast packets along with unicast.
4. Replicate servers to ensure high availability and avoid single point of failure.
5. Implement Gigabit or more network capacity connectivity to servers to avoid congestion and choke points.
6. Employ peer-to-peer (P2P) distributed nodes operating in an overlay network with multicast functionality to store content fragments and process search queries, and enable DE content delivery via existing wireless infrastructure at least cost.

Option 1 is impractical because the PSTN was provisioned several decades ago and its business model, featuring a tight budget, does not make it feasible for the government and private service providers of developing countries to upgrade its vast infrastructure [32], [33]. Option 2 is impractical because cellular service providers have been desiring to implement 3G and 4G services, but due to cost and government regulations they have not been able to do so [20]. Option 3 is complex, expensive and would not be scalable [34]. Option 4 also is expensive and would not address problems surrounding network congestion and bottlenecks or choke points [35], [36]. Option 5 would address bottleneck problems of servers, but calls for implementation of option 4 as well, which would also make the solution very expensive.

Option 6 could be the best, but it is complex, because the design of an architecture using a suitable emerging P2P system and technologies with a critical combination of parameters, properly calibrated metrics, that is scalable, efficient, and cost-effective is not trivial. This is because each system and technology has its own strength and weaknesses. There are P2P systems that handle search queries well but cannot address issues with network congestion and user node choke points. Then there are systems that try to minimize network congestion and bottlenecks, but cannot resolve problems surrounding single point of failure and are not scalable. There are also solutions that minimize nodes with single point of failure vulnerabilities, but are not resilient under *churn*. This occurs when there is frequent arrival and departure of peers in the network and it affects the resiliency of the routing, an inherent property of P2P systems [37]. Churn is a property that figures heavily in my research.

Overlay network construction is complex. There are protocols that build the overlay, which are tree-based and there are others that build mesh-based overlays. Tree-based overlays are more efficient and robust than mesh-based ones but can easily break the tree structure when non-leaf nodes leave the network. Mesh based overlay networks do not encounter this problem because there are redundant paths to other peers to quickly take over operations when a node leaves the network ungracefully, but this option calls for greater peer and network resources.

Considering the sixth option as the most viable, the goal of my thesis was to investigate P2P systems that work in an overlay network to handle efficient search queries in a distributed setting. This sets the stage of the hypothesis for the design of a cost-effective scalable architecture with P2P distributed nodes in an overlay to facilitate practical content delivery and exchange over emerging terrestrial limited capacity wireless networks.

1.5 Contributions and Summary

The most significant contribution of this work lies in discovering and integrating an effective architecture based on extensive research of emerging DE and Overlay Networking technologies, and P2P systems. Three major primary contributions and seven sub-contributions stem from the primary contributions of this thesis.

1.5.1 Contributions

Primary contributions are the following:

- Evaluated several P2P infrastructures and systems to make sure that none of the existing as-is designs meets the requirements of this work
    - To do this, I analyzed 18 P2P architectures and systems.
- Surveyed emerging P2P and Overlay technologies

    - Reviewed 12 protocols, routing algorithms, overlay tree structures and replication strategies to determine which ones may be good candidate(s) for the target architecture.

- Designed a novel architecture using emerging technologies for content delivery and exchange via limited capacity networks
    - This is a complex task as there are several P2P systems and protocols, each with its own advantages and disadvantages.
    - It proved difficult but possible to determine the parameters for an efficient and scalable architecture under churn in limited capacity networks.

Secondary contributions are the following:

- Surveyed DE tools, client-server architecture and data communications model
    - Surveyed popular commercial and open source learning management systems (LMSs), and synchronous distance education tools (SDETs) to select a suitable cost-effective tool to use with the target architecture.
    - Analyzed and simulated client-server architecture drawbacks to validate it is not a suitable candidate for the target architecture.
    - Reviewed problems with the traditional data communications model to justify it is not suitable for this work.

- Designed a framework for the target architecture and the four layer architecture
    - The target architecture framework is presented with a real world example and illustrations.

    - A four layer target architecture is designed and presented.

- Developed a top-down ordered structure for the simulation tests
    - Designed a top-down ordered structure of simulation experiments to describe test methodologies.
    - Simulated over 200 tests using different parameters and values, for analysis to conclude on design of the selected architecture.

1.5.2 Summary

This section summarizes contents of the remaining chapters of this thesis. My goal of integrating multiple technologies means an unusual level of review in multiple categories of existing technologies and previous work. Chapter 2 is a review of 18 structured and unstructured P2P systems and architectures employed by the entertainment and software industry. It concludes that none may be used as-is to deliver and exchange content over limited capacity networks. Chapter 3 looks at prior work of 12 emerging P2P and Overlay technologies that include overlay tree building protocols, search algorithms and replication strategies, to determine that none alone satisfies the requirements of this work. Chapter 4 is a comparison study on advantages and disadvantages of six LMSs and five SDETs. Problems with the existing communications model are presented in Chapter 5 where issues with the client-server model, IP multicast and communications media are described in depth. Chapter 5 also validates the drawbacks of the client-server model via simulated results. Chapter 6 is a discussion of requirements of the target architecture with justifications of each requirement. The framework of the target architecture with examples and illustrations is presented in chapter 7 that sets the stage for chapter 8. A top-down ordered structure of simulated experiments, describing each test methodology via an illustration is included in chapter 8. The simulated results support comparison of many parameters and metrics to determine the best solution for the target architecture toward the final design of the selected architecture of this work. An *n*-tier hierarchical training model is also presented with examples and a diagram in chapter 8. Conclusions and future directions are discussed in chapter 9.

CHAPTER 2: LITERATURE REVIEW OF P2P SYSTEMS AND ARCHITECTURES

2.1 Introduction

My research focuses on P2P systems to build on their positive traits of distributed nodes. This is to avoid single points of failure in the network for high availability, fast search and retrieval process, and low implementation and maintenance costs. Traditional client-server architectures lack these advantages due to their dependence on a central source for content storage and retrieval. Many P2P systems, architectures, and technologies have evolved over the years. Each has its own merits and one is better than the other as described in section 2.1 and chapter 3. DE is growing rapidly and its demand is huge. Students with smart devices using limited capacity Internet connections will soon be the major users of DE. Several P2P systems and architectures were surveyed to determine the best one for DE and general content delivery and exchange via the Internet that meets the dynamic needs of the learning landscape. Eighteen purely decentralized, hybrid decentralized and partially centralized, and structured and unstructured systems and architectures representing the primary P2P systems in use today are presented in this chapter. These systems primarily have been used to exchange data, music, video and software.

2.2 P2P Systems and Architectures

P2P systems and architectures have been used for a variety of applications such as communication and collaboration, distributed computation, Internet service support, database systems and content distribution [38]. They reduce collaboration costs and speed up communication via ad hoc administration of working groups [39]. P2P systems need to be fault tolerant, and be able to self-organize to maintain acceptable connectivity and performance, largely because they have a good number of transient nodes searching for content stored in other nodes in the absence of a *central server* (CS). P2P systems in general share files among a group either directly or via a distributed storage medium that is dynamically created. The dynamically assigned nodes, also known as *supernodes* (SN) assume the level of responsibility of a local central index to assist other peers in the network locate files they are searching by simply providing pointers to the files. A SN is similar to a peer node but is a more powerful machine with better network capacity links. The role of a CS is more elaborate and complex because it maintains directories of metadata describing the files that are stored by the peers in the network. The server does not store and distribute the content to the peers searching the file as is the case in a client-server architecture. SNs are not single points of failure because the network replaces them when they fail. CSs cannot be replaced automatically since they are not dynamically assigned by the network and hence represent a single point failure [38].

Androutsellis et al. [38] group P2P technologies into P2P applications and P2P infrastructures. P2P applications are responsible for setting up a network of peers to publish, store, and distribute content. P2P infrastructures are responsible for routing requests and messages, providing user anonymity, and handling reputation management. There are two types of P2P overlay network organizations: 1) *unstructured* and 2) *structured*.

2.3 Unstructured Systems

With unstructured systems, there is no relation between the overlay and files placed in nodes of the overlay network. These systems are more suitable for networks with highly transient nodes but require extensive search for locating files [38]. This type of network is not scalable, has a low level of fault tolerance, and cannot guarantee availability of the content at all times. Napster [38], Gnutella [40] and Kazaa [42] are good examples of unstructured systems. Here, nodes do not know which hosts may have the data being sought and are unable to forward the queries efficiently. There is unnecessary probing of hosts that do not have the content being queried. This is particularly apparent with Gnutella systems, which use a flooding algorithm to propagate the query.

2.3.1 Hybrid Decentralized

Hybrid decentralized [38] is one type of unstructured architecture that has a CS. This type of system is easy to implement and files can be located quickly. However, it is not scalable because there is limited storage and processing power of the CS, and it may become vulnerable to attacks and pose to be a single point of failure. This system is complex and expensive due to the management of CSs [38]. When a node wishes to join the network, it reports on files it maintains to the CS. The CS is queried by the *requesting node* (RN) for a file. The CS checks its index for a match and sends information on *node store* (NS) peers having that file to the RN, and RN directly contacts the NS(s) to download the file. Napster and Publius [43] systems belong to the hybrid decentralized architecture. They do not have the option of replacing or adding CSs [43]. The architecture and search process is illustrated in Figure 2.

A simple example is included here to help the reader understand the mechanics of this system. In Figure 2, Node 2 needs a file and in step 1 it queries the CS. In step 2, CS refers to its index for a match. It finds a match and sees Node 1 has the file. CS replies to Node 2 in step 3 with this information. In step 4, Node 2 contacts Node 1 directly and requests file download. Node 1 in step 5 sends the file to Node 2.

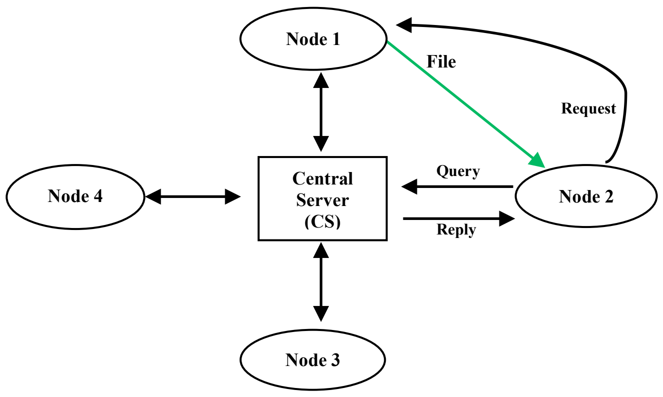

Figure 2: Hybrid decentralized architecture

2.3.2 Purely Decentralized

The purely decentralized architecture [38] does not have a CS and the virtual overlay network is created using routing mechanisms of its application software. There is direct communication between the peers without any sort of assistance of a CS or SN. Gnutella [40] and FreeHaven [44] are examples of purely decentralized unstructured systems where the search method is non-deterministic. Gnutella is described here because it provides sufficient illustration of this class of architecture.

The application level protocol of Gnutella uses four message types for group membership and file search. A *Ping* message is sent to its neighboring nodes by a node that joins the network to request IP and port information along with the number and size of files the nodes are sharing. A *Pong* message is the reply to the ping message. Ping and Pong are used for *group membership*. A *Query* message is a request with search string, and IP, port and minimum required speed information of the host with a match. A *Query Hits* message is the reply to Query message. Query and Query Hits are used for *search*. The Ping is flooded. It is first sent by the *joining node* (JN) to all nodes it has open Transmission Control Protocol (TCP) connections, and then the *neighboring nodes* (NN) propagate the Ping to their neighbors. To avoid infinite spread, each message has a *time-to-live* (TTL) counter field. To help avoid loops, each message also has a *unique identifier* to detect and drop duplicate packets. Reply message identifier is same as the original message, so the host refers to its routing table prior to forwarding the reply message to ensure it is sent over the same link the original message was received. Hosts maintain a dynamic routing table that is populated with message identifiers and node addresses to prevent re-broadcast of messages that were just sent. Flooding mechanism is a disadvantage since it leads to congestion and over utilization of network capacity. Gnutella uses two message types, *Get* and *Push*, for file *downloads*.

In Figure 3 (step 1 of this search), JN sends a query message to its NNs for a file. *Node 1* (N1) and *Node 3* (N3) check their respective tables for a match to the search string, and do not find one. They propagate the query to their respective neighbor *Node 2* (N2). Note, N1 and N3 do not back broadcast the query message to JN because they notice from their routing tables that the message identifier of the query was received from JN, so there is no point in sending it back to JN. In Figure 4 (step 2 of this search), we see N2 finds a match to the query message string and sends a query hit message in reply to the query message it just received. The query hit is forwarded by N1 and N3 to JN, and a direct connection is established between JN and N2 to initiate download. Solid lines represent network connections.

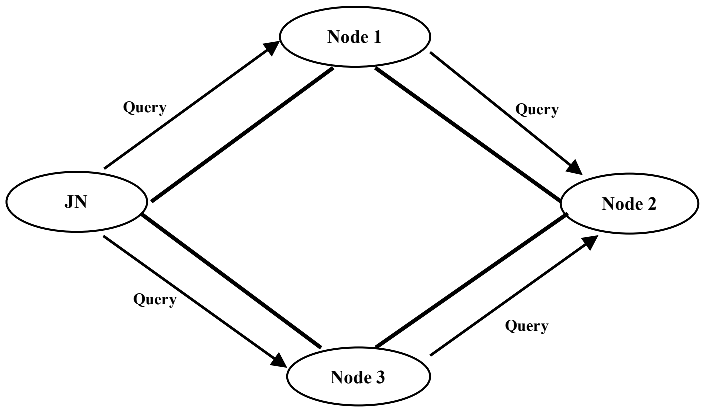

Figure 3: Purely decentralized architecture (Gnutella) search process - step 1

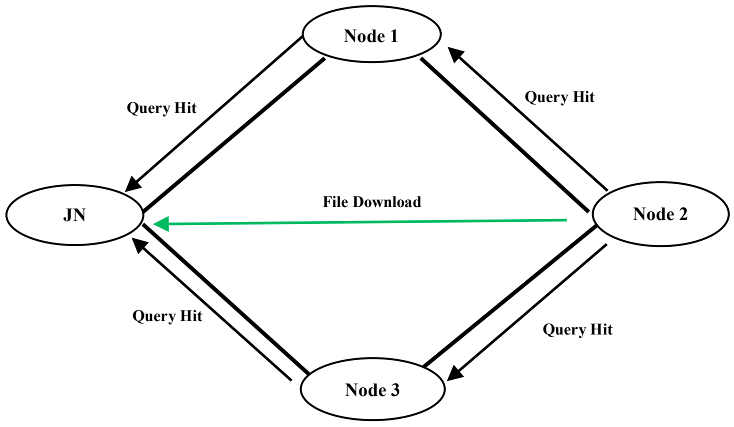

Figure 4: Purely decentralized architecture (Gnutella) search process- step 2

In this architecture, congestion is a major problem, and it is due to message flooding. Resiliency is another problem because nodes join and leave the network frequently. End-to-end (E2E) delay is significant since messages propagate through intermediate nodes that lookup their tables for a match or to forward the message onward. Performance degrades as the network size increases so this architecture is not scalable [45].

2.3.3 Partially Centralized

The partially centralized architecture [38] introduces the concept of SN. A SN is automatically elected if it has more processing power, memory and network capacity than other nodes. It maintains an index of files that is shared by peers, and act as their *proxies.* This is because search queries to the peers are processed by the SNs. Since the burden is carried by the SNs, this search method lessens discovery time and relieves general peers from processing load, and cuts network capacity usage. The SNs may be replicated to avoid having a single point of failure. This architecture has the organizational pattern of *power-law networks* [40] which makes it robust in comparison to purely decentralized architecture. It has many more links to SNs than it does to a normal peer. This is why SNs undertake heavier network load than the other peers. On the negative side, SNs are vulnerable, as they may become bottlenecks or choke points, especially during a flash crowd. Also, power law graphs tend to become segmented (tree breaks) when peers arrive and depart the network frequently [46]. Kazaa [42] and Edutella [47] are examples of partially centralized implementation. A later version of Gnutella [40], [48] adopted the concept of SNs to ease its search process. Perhaps, by having solutions in place to address these disadvantages, it would enable partially centralized system to be suited for design of the target architecture. Chapter 3 looks at emerging technologies for possible solutions to overcome the drawbacks of this architecture.

2.3.4 BitTorrent

BitTorrent (BT) is an unstructured P2P system that leverages a *tracker* server (TS) to resolve queries. The key feature and uniqueness of BT is its ability to enforce fairness in the network by preventing free riding. Peers or clients are categorized as *seeders or seeds* and *downloaders or leechers*. Seeds have all pieces of the file and stay in the system so other peers can download from them, and leechers have a few or none of the pieces of the file. However, there is no mechanism in place to encourage seeds to stay in the network for a long time, and they may leave at any time. Each file is broken down into small block sizes or *chunks* of 256 KB, and the blocks are further broken into 16 KB fixed size sub pieces. Sub pieces are there to allow pipelining in order to mask the request-response latency [49]. The entire file is typically a large file of 100 MB or more, and is stored by a seed. Leechers may have some blocks of the entire file and are looking for the missing blocks of the file in the network. Users cannot search for files/chunks on their own when they join the network. They need to first contact the web server that hosts *torrent* files. A torrent file has metadata about a file that includes the size of chunks and their checksums along with address information about the tracker [50]. Torrent files are created by users and uploaded to the tracker server. The tracker server is not active in distributing files; it just maintains list of active seeds with the file, and peers with blocks of the file.

When a new node joins the network, it contacts a standard *web server* (WS) for the torrent file. Once it receives the torrent with the URL of the TS, it contacts the TS, and gets a list of active seeds, and leechers that have the entire file or blocks of the entire file. Once it receives the list, it connects to those peers to find out which chunks each of those peers have, and requests the chunks it does not have. It also provides the chunks it has to the peers in the *swarm*. A swarm is the group of seeds and leechers that is sharing the file the new peer is searching for, and information about the nodes in the swarm is maintained in the cache of the peers in the swarm. Leechers also upload the blocks they have to other leechers in the swarm. This is a good practice since it discourages free riding because in BT, a peer will only serve a peer if that peer serves it. Uploading is initiated via a mechanism called *unchoking*, where a peer opens connections to four peers with the best/fast downloading rates at a given time. Once this is done, it concurrently uploads to those four peers. Four peers are chosen because it lets the built-in congestion control of TCP to saturate to its upload capacity. To avoid resource wastage due to rapid *choking* and unchoking, the peers decide who they wish to choke every ten seconds, and leave it as-is for another ten seconds. This is because ten seconds is ample time for TCP to have new transfers at their full capacity [46]. Choking is where a peer closes connection with another peer.

Every 30 seconds, a peer runs an *optimistic unchoking* algorithm, when it uploads to the fifth peer with the fastest download rate that is randomly selected from the list of peers that are requesting downloads. This is done because although it assumes it is uploading to four peers with the best downloading rates, there may be new peers added to the network that are not downloading, and may have better downloading capacity. Plus, it gives an opportunity to the new peer that just joined the system and has nothing to offer to download the first block which it may then upload to other peers. With optimistic unchoking, the maximum number of *downloading peers* (DP) at a given time is five. Since optimistic unchoking is initiated every 30 seconds, the peer has to drop a peer with the lowest download rate to keep the maximum number of DPs to five during a specific time. A similar mechanism is used by the DPs but here the opposite takes place, where four peers with best uploading rates are selected. Every 10 seconds the DP checks transfer speed on a moving average of 20 seconds and chokes the *uploading peers* (UP) with worst uploading rate and replaces UP with the better uploading rate seed [46], [49].

BT peers do not have a global capacity view. To ensure uniform distribution of chunks in all peers, *rarest-first policy* technique is used where rare chunks are downloaded first. It uses *endgame mode* mechanism at the end of a file download, so that a peer with most of the chunks of a file does not have to wait for a long period to download remaining few chunks from a peer with slow transfer rates that has those chunks, by being able to query all peers for the missing few remaining chunks. One of the disadvantages with this architecture is the TS because it may be vulnerable to single point of failure. Replication of TS would address this issue because they are not overwhelmed to process file sharing requests as servers of client server architecture. Torrents are very useful for fast lookups. Choking and unchoking mechanisms are great for fast and efficient downloads. The use of tit-for-tat to discourage free riding is another positive trait of BT.

2.4 Structured Systems

Due to the scalability problems of unstructured systems, researchers introduced structured systems where files are placed in precise locations and the overlay is tightly controlled. Here, distributed routing tables maintain information on content using file identifiers and node address. If the query string is based on the identifier there will be an exact match, which is why structured systems are considered to be scalable, because unknown searches are reduced. However, there is the problem of maintaining this type of system in presence of transient node population [38], [51]. Chord [45], CAN [52], PAST [53] and Tapestry [54] are examples of structured systems.

There is also a hybrid version known as a loosely structured system, where exact location of the content is not clearly specified as it is with structured systems, but offers routing hints to significantly ease a blind search. Freenet [55], [56] is an example of loosely structured system. The architecture of the Freenet system is presented in the next section.

2.4.1 Freenet- Hybrid Architecture

Freenet is a decentralized loosely structured system that uses key identifiers to estimate file location via a process known as chain mode propagation to forward queries from node-to-node. Unique binary keys are used to identify the files and a timeout (hops-to-live) value is assigned to four message types that include unique message identifiers. The identifiers are used to detect duplicate messages and avoid loops. The hash value of the descriptive text string describing the file is the key [55]. As a notional example, descriptive text string “distance/education” is hashed to obtain hash value “AB4CD1” as the file key to insert and query. This hash value is just an example, intended to illustrate the concepts of Freenet.

To *insert* data, a user goes through the following steps. Step 1 involves hashing of the descriptive string to obtain the file key, storing it in user’s node or RN, and assigning a hops-to-live field to specify how many nodes will store this. In step 2, an insert proposal is sent to the NN. The NN in step 3 checks its own store to see whether the key exists. If it does, NN sends the pre-existing file to RN as if the request was made for it. Upon receiving this file, RN detects a collision has occurred and hashes a different descriptive text string for the file, and sends it back. In step 4, NN again checks its own store for the key and if it is not found, it refers to its routing table to locate the *corresponding node* (CN) that has a similar key determined by lexicographic distance, and forwards it to the CN. The process continues until all nodes per hops-to-live number are reached without a collision. An *all clear* message is then sent back to the insert RN in step 5. In step 6, RN sends the data to store to nodes up the chain that just inserted the file key. This mechanism controls search breadth by ensuring new data is stored in nodes with similar keys, thus storing similar files. The other advantage with this is a new node is able to inform other nodes of its arrival when it inserts data.

The *search* mechanism of this architecture is described next. Retrieving data follows steps similar to those used in data insert. The user hashes a descriptive text string to obtain the file key and sends a *request* message with the key and hops-to-live value to the user's node or RN. When a NN receives the request message, it checks for the file and if found, it is sent to the RN. Otherwise, it is forwarded to the CN with similar key. This process continues until there is a match and the data is sent back upstream to the RN. As it propagates to the RN, all nodes in the chain make a copy of the data in cache and update the routing table referencing the original data source with requested key value. This shortens the search when there is a request for this file at a later time.

Figure 5 illustrates the search process with an example. The RN hashes a descriptive text to obtain a file key, and sends a *request* message with the key and hops-to-live value to N1. N1 checks to see if it has the file, does not find it, and refers to its routing table to forward it to the CN that has a similar key value. N1 forwards it to N2, which checks its store and does not find the file. It tries to forward the request onward but is unable to do so, and sends a *fail* message to N1.

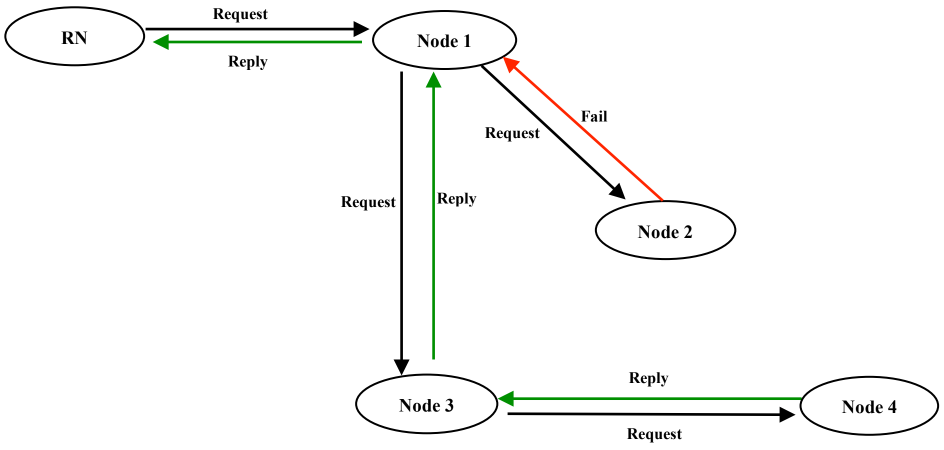

Figure 5: Loosely structured architecture (Freenet) search process

Upon receipt of the failed message, N1 refers back to its routing table for the CN with the second nearest key value to forward the request message. N3 turns out to be the second choice. N1 then forwards the request message to N3 and it checks for the file, does not find it and notes Node 4 (N4) from its routing table to be the CN to forward the message onward. N4 has the data. It sends it to N3 which sends it to N1, which sends it to RN. In the process, N3 and N1 cache the data, and update their respective routing table with requested key and N4 as the data source.

Freenet is complex and is slow to adapt as transient peers join and leave the network, because it is a cooperative distributed file system and due to its chain mode propagation structure. It provides some degree of anonymity, but does not guarantee document retrieval and low bound retrieval cost [45].

2.4.2 Chord Architecture

Chord [45] is a structured system that maps file identifiers onto node identifiers. It does so by assigning keys to nodes with consistent hashing to identify data files in nodes. Consistent hashing help balance loads because nearly the same number of keys is distributed to each node, leading to little movement of keys as nodes join and leave the network. It scales well since a node does not need to have information of all nodes in the network in its routing table. In an *N-node* system, each node maintains information about $O(\log N)$ other nodes in its routing table, and is able to resolve queries via $O(\log N)$ messages to other nodes. As number of nodes increase exponentially, search time increases linearly. Each node and key is assigned a *m*-bit *identifier* using Secure Hash Algorithm-1 (SHA-1) hash function. The node IP address is hashed to obtain node identifier and key is hashed to obtain key identifier. A Chord ring is an i*dentifier circle* where identifiers are ordered modulo 2^m. Key identifier k is assigned to the first node whose node identifier n is equal to k. If there is no equal value of n to k, it follows k on the circle to the next n of value greater than k. The n where k is stored is known as the *successor node* of k denoted by *successor(k)*. Identifiers are on the circle as numbers from 0 to $2^m - 1$. First *successor(k)* node is clockwise of k. Refer to Figure 6 for an example.

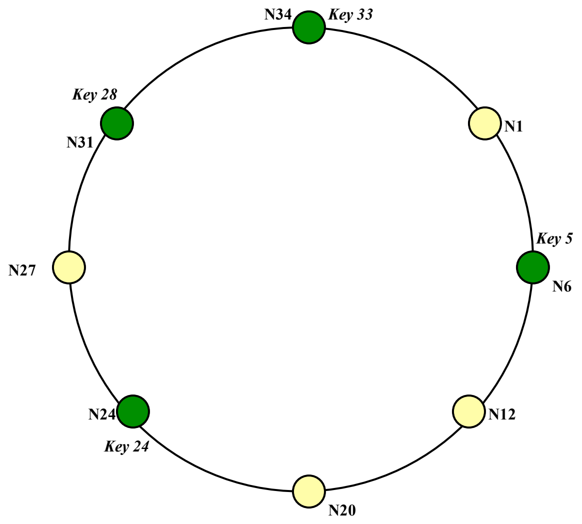

Figure 6: Structured architecture - Chord system

The Chord ring in Figure 6 has *m=4* and has 8 nodes storing 4 keys. The successor of *k5* is N6, *k24* is N24, *k28* is N31 and *k33* is N34. Due to consistent hashing, there is not much disruption as nodes join and leave the network, but hashing uses additional computer resources. The consistent hashing map is maintained by assigning keys assigned to *n's* successor to node *n* that joins the network. If *n* leaves the network, its keys would be assigned back to *n's* successor node. For example, if a node with identifier 30 joins the network, it would capture *k28* from N31. This information is maintained in a routing table with *m* entries in each node called the *finger table*. The i^{th} entry in the table of node *n* identifies the first node *s* that succeeds *n* by at least 2^{i-1} on the circle, and *s=successor* derived from $(n+2^{i-1}) \bmod 2^m$ where $1 \leq i \leq m$. The finger table of N6 of Chord ring in Figure 6 is illustrated in Table 4. First finger of N6 points to N7 because N7 is the first node that succeeds $(6+2^0) \bmod 2^4 = (6+1) \bmod 16 = 7$. Table 4 has remaining entries of the finger table.

Table 4: Finger table of node 6 of Chord ring in Figure 6

Node *n's* i^{th} entry	*Successor (s)*
N6+*1*	*N7*
N6+*2*	*N8*
N6+*4*	*N10*
N6+*8*	*N14*

There are two important points to consider with this approach. Firstly, each node has information about a small number of nodes that are closely following it on the identifier circle, and does not have information of nodes farther away. Secondly, it does not have information of successors for all keys.

For example, N6 cannot determine the successor of *k28* because the entry is not in its finger table. If *k* is between *n* and *s*, search is complete and *n* returns its *s*. If the value is not found in that range, *n* searches to the node *n'* that immediately precedes *k* and invokes algorithm to find *s* of *n'*. For example, in Figure 6 say N6 wants to find the successor of *k24* and because the largest finger of N6 preceding *k24* is *N14* in Table 4, it will turn to *N14* to search and resolve the query.

The algorithm of this process is shown in Figure 7 as described in [45].

```
// request n to find successor of k

n.find_successor (k)
 if (k ϵ (n,successor))
  return successor;
 else
  n' = closest_preceding_node (k);
  return n'.find_successor (k);


// search routing table for highest predecessor of k

n.closest_preceding_node (k)
 for i = m downto 1
  if (finger [i] ϵ (n,k))
   return finger [i];
 return n;
```

Figure 7: Chord key lookup algorithm [45]

Performance of Chord degrades slowly as nodes join and leave the network, and routing information is no longer current. Since the overlay is based on the underlying physical IP network, there may be several failures in a Chord network during a single failure in the IP network [57].

Although the finger table provides useful information to locate *k* when a node fails or leaves the overlay, delivery ratio may deteriorate when there are transient nodes with low session time and high failure rates. This is a problem with Chord architecture because it is a structured system where the overlay network is tightly controlled. Exact match lookup is another major disadvantage of structured systems. These result in negative impact on availability and fault tolerance [57]. Chord also does not make distinct effort to attain good network locality [53].

2.4.3 CAN Architecture

Design of the Content Addressable Network (CAN) [52] is based on *d*-Dimension Cartesian coordinate space that stores (*Key k*, *Value v*) pairs. It implements a distributed hash table (DHT) to map *k* onto *v*. That is, it maps a file name to its location in the network. It also supports insertion, lookup and deletion of (*k, v*) pairs in the table [38]. A portion of the hash table (*zone*) and information about few adjacent zones in the table are stored in each node of the CAN network. When there is request to insert, lookup or delete a particular key, the request is routed to the node maintaining the zone with the key. The entire coordinate space is dynamically partitioned between the nodes in the system so that each node has its own zone. The *k* has to be deterministically mapped to a point *p* on the coordinate space using a hash function to store the (*Key k*, *Value v*) pair at the node that owns the zone where *p* lies. A node may find an entry corresponding to *k* by applying the hash function to map *k* onto *p* to retrieve corresponding *v* from the node covering *p.* If the node does not own the zone that covers *p,* the request has to be routed onward until it reaches the node with the zone covering *p.*

The routing table in CAN nodes include IP addresses of nodes with immediate neighbor zones in the coordinate space. This information is used to route requests between arbitrary points in space. When a new node joins the network it randomly chooses a *p* and sends a *join* request to the node covering *p.* It is allocated a portion of the coordinate space by splitting the zone of the existing node covering *p* to which *join* request was sent. Zones occupied by a node with hash table entries are transferred to one of the neighboring nodes once the node leaves the CAN. Node status is monitored via periodic updates that are sent by each node to its neighbors. The update message has the node's zone coordinates, list of neighbors and their coordinates. If an update is not received for a prolonged period of time, it is assumed that node failed and a takeover mechanism is initiated. Network latency and fault tolerance shortcomings of CAN are addressed in [52] by proposing advanced routing metrics, allowing multiple nodes to share the same zone, mapping the same *k* onto different points, and using caching and replication techniques. CAN appear to have safeguards in place to handle node failures but the routing mechanism is complex. The extra maintenance protocol of CAN to remap the identifier space onto nodes requires additional resources and time [45]. Its lookup cost is $O(dN^{1/d})$ because each node maintains $O(d)$ state. This is high in comparison to $O(\log N)$ because CAN node's state does not depend on N, and lookup cost increases faster than $\log N$ [52], [53]. This increases delay in processing search requests.

2.4.4 Tapestry Architecture

Tapestry [54] is a distributed, self-administering and fault tolerant system that locates objects, and route messages to them. It is stable because it bypasses failed nodes and routes, and can adapt to changes in the topology. Tapestry's nodes come and go, and self-organize to build the topology. It maintains routing and location information in nodes distributed throughout the network, and is able to rebuild or refresh network topology as it detects failure. The concept of object location and routing mechanisms presented by the *Plaxton mesh* in [58] is used in Tapestry. This uses small and constant size routing maps, to locate objects and route messages to them over an arbitrarily sized overlay. Each node maintains a *neighbor map* that has multiple levels. Each level l has pointers to nodes with ID matching l digits. A level has several entries equal to the base of the ID. The i^{th} entry in the l^{th} level points to the closest node whose ID matches up to a digit position of the number in the neighbor map. There is a root node for each object, which guarantees object location. Since the Plaxton mesh is based on static nodes, Tapestry's design is extended to handle a transient population of nodes and means to adapt to changes in the network, and maintain fault tolerance [38].

Other optimizations also are implemented in Tapestry as described in [54]. Every node maintains a list of back-pointers that point to neighbor nodes, which is useful for updates to new nodes. Content is cached in nodes to recover from routing failure or object location. Plus, the neighbor map maintains two backup neighbors in case the closest neighbor fails. To address single point of failure, Tapestry assigns multiple roots to each object, but this is resource intensive. It is able to improve performance of the nodes by tuning neighbor pointers that updates latency values. This process detects query hotspots and resolves latency issue by locating other nodes with less latency where objects may be stored to improve response time. This approach becomes ineffective when there are many peers with low session times in the network. Oceanstore [59] uses Tapestry where it enhances performance and fault tolerance by adding another layer. Here, nodes are able to adapt to outages and denial of service attacks by monitoring usage patterns, network activity and resource availability [38]. Root replication is a good approach to counter single point of failure but has to be done efficiently to avoid generating a large number of control messages. Tapestry does not describe any mechanism in place to address congestion and delivery ratio. It attempts to reduce routing latency but at the cost of a complex join protocol that need to initialize the routing table of the new node based on the information received from the join message of the nodes traversed [45]. Also, its performance deteriorates during heavy churn and when the rate of peer failures goes up.

2.4.5 Structured Architecture Problems

P2P networks have transient node populations and it is costly for structured systems to maintain routing tables in this type of environment, especially if the nodes have short session times. The other major disadvantage resulting from use of structured systems is the support of exact match lookups because the exact identifier *k* of the data item must be known to locate the node that stores it [38]. This is not trivial.

2.5 Two-tier Overlay Multicast Architecture (TOMA)

Lao et al. [60] propose a Two-tier Overlay Multicast Architecture (TOMA) and a lightweight, scalable protocol called OLAMP (Overlay Aggregated Multicast Protocol) that allows multiple groups to share one delivery tree. It reduces the number of multicast trees in the domain, which reduces tree management overhead that lessens multicast state information stored in routers [60].

To address challenges of IP multicast, the researchers propose TOMA to provide scalable and efficient multicast support for various group communication applications. In this architecture, Multicast Service Overlay Network (MSON) is presented as the backbone service domain. The MSON has service nodes or proxies strategically deployed by an MSON Internet Service Provider (ISP). The MSON provider provisions its overlay network based on user traffic characteristics, buys capacity from Tier 1 network service providers and sells multicast services to group coordinators. Provisioning the overlay require long term monitoring of the network that is extra work for the provider. Outside MSON, end hosts subscribe to MSON by connecting to proxies of the MSON provider and form clusters around these proxies. For efficient data delivery, application layer multicast is used in each cluster. The end users just pay for their network connection service outside MSON [60].

Efficient MSON management and provisioning are critical issues to consider. Since MSON has to accommodate a large number of multicast groups, it is important for a MSON provider to efficiently establish and manage several multicast trees. Also, based on user traffic characteristics, the MSON provider needs to provision the overlay network carefully to reduce cost and improve service quality. Outside MSON, an efficient proxy selection mechanism is presented. For data transmission inside clusters, a core-based application layer multicast routing approach is used. Aggregated multicast is used to allow multiple groups with similar members to share a single delivery tree that helps reduce the number of multicast trees in the domain to relieve tree management overhead. This is possible because it cuts multicast state information stored in the routers.

In access networks outside the MSON, multicast is used by peers in the same cluster along with a member proxy of that cluster. Nodes connect to the same member proxy from a cluster that is at the edge of the MSON. Nodes in the cluster communicate with each other and update path quality information. They also report this to their member proxy that stores group information of nodes in the cluster to construct multicast trees. The member proxy repairs the multicast tree as peers ungracefully leave the cluster.

TOMA has been compared with NICE [61] and Narada [62]. Experimental results show that TOMA's performance is better than NICE and Narada. This is true in all cases and particularly when the group size increases. This is due to two reasons. Firstly, TOMA's ability to provision the overlay network to resemble the underlying topology plays a major role in improving performance. In contrast, NICE and Narada rely on path probing techniques that leads to construction of overlay topologies of poor quality. Secondly, by using proxies as intermediate nodes, packets can be replicated at proxies that reduce the number of redundant packets sent over physical links.

The number of links on the path from source to a member is known as path length. On average, NICE and Narada have longer path lengths than TOMA. Better performance is achieved by TOMA factoring in link stress. This is due to efficient construction of the overlay.

TOMA has a lower number of control messages than Narada and NICE because local clusters are in most part static, and a member stays in the same cluster regardless of member movement. This architecture scales well, but control messages to maintain trees and groups increase as group sizes become large. Due to a large number of multicast groups, tree management can be complex for the MSON provider. OLAMP is proposed for multiple groups to share one delivery tree, but its not a complete solution. This is a major problem for end systems with limited network capacity to handle large number of control messages, which lead to greater latency. Next, there may be multiple failures in the overlay network during a single failure in the IP network because it is based on the underlying physical IP infrastructure. The other concern is it is unlikely TOMA actually will be supported by ISPs in most locations because it is expensive to provision.

2.6 Reliable Overlay Multicast Architecture (ROMA)

Kwon et al. [63] present ROMA, a TCP-based delivery architecture that avoids limitations of varying transmission nodes. It performs well with multiple stream rates, where individual rates can match end-to-end available capacity along the path. This is possible using small buffers at application level relays and TCP protocol. Here, a forward-when-feasible approach is used instead of a traditional store-and-forward mechanism. Each intermediary only forwards received packets to downstream hosts that can immediately write to its TCP socket. It avoids tightly coupled TCP connections at intermediate end-systems that limit performance by reducing transfer rates in the system to the slowest receiver rate. Reliable delivery of packets is ensured using an erasure-resilient code technique that has been widely used for reliable IP multicast. This however imposes load on the peers and may deteriorate performance.

The Kwon et al. paper discusses the effects on performance when TCP chains develop with their approach. The chains form from the sender to end-hosts on a ROMA overlay. Overlay multicast typically has performance issues due to factors such as link stress, suboptimal routes, increased latency, and end-host packet processing. However, TCP chains improve performance by finding an alternative wider overlay path. The narrowest hop of this path gives better TCP throughput than default IP path. ROMA is able to accomplish this by maximizing minimum expected TCP throughput of the multi-hop path. In this architecture, nodes send traditional Acknowledgement (ACK) packets to acknowledge successful receipt of packets. This is a major disadvantage in limited capacity networks because it causes congestion due to excess control overhead.

2.7 Overcast Architecture

Jannotti et al. [64] present the Overcast multicast system that may be incrementally deployed on the Internet. It is scalable and offers reliable single source multicast. It uses a simple protocol to build data distribution trees for adapting to network changes. Overcast is able to support fast joins. It does so with its protocol that tracks global state of changing distribution trees.

An Overcast system is an overlay network that has a central source. This may be replicated for fault tolerance, but the concept of a central source always has the vulnerability of single point of failure. The network has a number of internal Overcast nodes and clients throughout a network fabric. It is able to organize internal nodes into a distribution tree rooted at the source using a tree building protocol. Changes in underlying network fabric are adapted by the tree building protocol. Overcast is able to provide large scale, reliable multicast groups using distribution trees that are built with a tree-building protocol to create high capacity channels from source to all nodes. These are suitable for on-demand and live data delivery [64]. Since Overcast is an overlay network and is not able to optimize itself without good knowledge of the substrate network topology, optimizing for all paths becomes an issue [62].

Overcast uses a protocol called Up/Down to keep track of all nodes in the network. Up or down status of nodes is reported by this protocol to the root. This process makes Overcast networks efficient and productive, but causes congestion during frequent peer outages. Since the root is a single point of failure, Overcast addresses this vulnerability by having Domain Name System (DNS) name of the root resolve to any number of replicated roots in a round robin manner. There is a non-root sender active at any particular time. This sender sends unicast messages to the root, which then sends received messages via true multicast on behalf of the sender to other nodes. Other projects [65], [66] have used this approach.

Overcast addresses efficient tree-building mechanism but does not take into account instabilities of the Internet caused by transient nodes which may contribute to low delivery ratio and unacceptable end-to-end latency.

2.8 Narada Brokering Architecture

Pallickara et al. [67] uses the publish/subscribe paradigm and presents a mechanism to discover brokers in distributed messaging infrastructures. Traditional solutions using simple methods to access remote brokers via another entity may lead to capacity degradations and inefficient use of newly added brokers. The publish/subscribe paradigm addresses this deficiency, because it offers decoupling of the message producer and consumer roles that may be present in interacting entities/services. This architecture uses message oriented middleware (MoM) [68], [69] that routes messages from the publisher to the subscriber. It ensures the right content is properly routed from the producer to correct consumers. A subscriber registers its areas of interest by subscribing to topics. When a publisher posts a specific topic, the MoM routes this event to the subscriber that registered with an interest for that topic.

The system proposed by the researchers, NaradaBrokering (NB) [68]–[74], a distributed messaging infrastructure leveraging the publish/subscribe paradigm, was used for experimentation to characterize NB. It provides a MoM that makes communications between entities such as clients, resources, services and proxies simple and transparent, and offers a notification framework [71], that allows routing of messages from the originators to only registered consumers with similar interest.

A broker network has many cooperating brokers. They work together to provide access to hosted services and functionalities. An entity first needs to connect to a broker after which it has access to a wide variety of services. In a similar fashion, an entity may also add a broker to this network. In both cases, it is necessary for the entity to discover a broker but there is a problem in discovering brokers. This is due to two key reasons. Firstly, the discovery process must return a valid and non-empty set of brokers. Secondly, brokers that are within this subset have to maximize the added entity's ability to access and negotiate accesses to services that are hosted on the broker network [67].

In a dynamic brokering environment where NB operates, it is common for brokers to join and leave a broker network. A solution for this is needed. The discovery process has to be dynamic based on the current state of the broker network and needs to factor in its own failures and unexpected faulty behaviors. Broker Discovery Nodes (BDNs) are introduced; these are specialized and registered nodes within the broker network. They help with the discovery of brokers by maintaining information regarding broker nodes within the system. When a broker is configured, it advertises and registers its presence with one or more BDNs. A new node arriving into the system sends a broker discovery request that is forwarded to a publicly known BDN. Either TCP or User Datagram Protocol (UDP) transport protocols may be used to communicate with the BDN. Once the BDN receives the request, it propagates the request through the broker network. When a broker decides to respond to a *broker discovery request*, it builds a *broker discovery response*. The RN receives the responses from one or more brokers in the broker network and creates a subset of target brokers. It then selects the broker it wishes to connect to from this subset. The target broker is selected based on *delay metric* measured via ping requests [67].

Experimental results show most time is spent on waiting for the initial response. As the number of brokers increases, this gets worse. To address this, a timeout period is introduced to make better decisions on which broker to connect with. This problem is somewhat addressed by multicasting the request so the brokers in local networks receive the request directly rather than having it forwarded by some broker discovery node. However, multicast may not work if some brokers listen on a different multicast address or port or if multicast service is disabled. Timeout and multicast are good approaches but they are not new technology, and do not offer innovative contribution.

The *broker discovery response* has information on total memory used, number of links the broker is connected to and Central Processing Unit (CPU) load of the broker, which is useful for better routing. NB uses UDP as the communication protocol to transmit broker discovery response and does not have any mechanism in place to detect lost packets. This architecture has similar properties of the learned model [75] without file cluster concept. This architecture has good routing mechanism but does not have good measures in place to be resilient under churn.

2.9 Distance Education System Architecture

There are several DE system architectures but this focus more on the application than delivery. An architecture that includes some aspects of a delivery mechanism is presented in this section. There are no DE architectures available that are specific to type of content delivery.

A service oriented e-assessment architecture is presented in [76] that leverages cross domain use cases to provide flexible access to different assessment systems. Colleges and departments in universities tend to have several disparate online assessment systems because they implement their own versions to meet their domain needs. This architecture addresses that problem because it provides one e-assessment system for use across colleges and departments in a university. The researchers present this architecture for Modular Assessment System for Modern Learning Settings (MASS) that has the following features: 1) flexible design, 2) user friendly interfaces, 3) assessment environment for various learning and assessment settings, 4) management and semi-automatic support over the assessment life cycle from exercise creation and storage to grading and feedback, 5) rubrics design, 6) support for various educational objectives and subjects, 7) results analysis and feedback provision, 8) standard-conform information and services to be sharable, reusable and interoperable that include test questions and answers, and 9) security and privacy [76].

Al-Smadi and Gutl [76] state MASS is a SOA-based assessment system that integrates several assessment tools for assessing different application domains, and depends on interoperable cross-domain web services to extend its core services.

The architecture has four tiers: 1) client application tier, 2) business tier, 3) service tier and 4) resource tier.

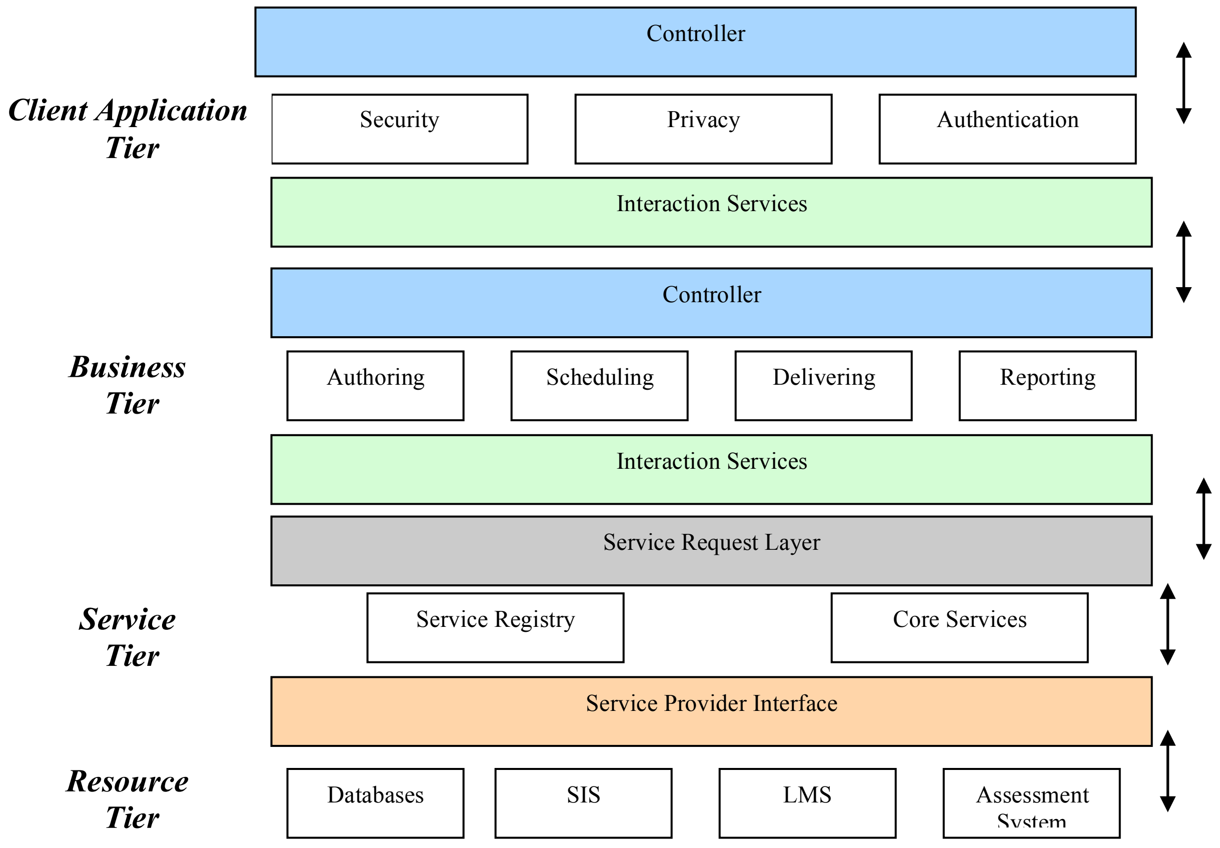

Figure 8: MASS architecture for e-assessment [76]

Figure 8 illustrates this architecture. In the client application tier, user agents interact with the system and business tier using technologies such as Simple Object Access Protocol (SOAP) and Extensible Markup Language (XML)- Remote Procedure Call (RPC). Business logic is in the business tier, which has MASS modules, and there are sets of core services for each module in the service layer.

In the service tier, service providers have agents to provide service by publishing services on services registry, and service client agents find service descriptions on the services registry and request execution of services. To locate a service, service requestor invokes a web service using the binding information in the requested service description located in the services registry. Service providers are domain related and are located in the resource tier that provides their business logic services, such as grading a math problem, as web services to the services registry, using the service provider interface. Service invocation is handled in the service request layer by searching for the requested service in core services. If the service is not found, it searches the service registry for new registered services. Once a new registered service from service registry is used by the business tier module, it becomes available in core services module so the next time there is a request for this web service it can be accessed directly from core services. The resource tier has MASS databases, domain related system such as assessment system, business related systems such as Student Information System (SIS) and LMS. There is a set of web services for each of these systems and resources [76].

This architecture may be used as a component of the target architecture to integrate several assessment tools for assessing different application domains. It is not, however, a solution for content delivery across networks.

2.10 Summary of Architectures

In the presence of heavy churn where peers have low session times and high failure rates, it is necessary for the architecture to adapt quickly to changes with minimal disruption. Structured architectures do not fare well in such environments. Chord, CAN and Tapestry are structured systems with efficient search mechanisms, but the overlay is tightly coupled because files are strategically placed in the network. This makes them vulnerable to failure as nodes join and leave the network in short intervals. Chord uses a finger table, and CAN send periodic updates to detect node failures and initiate takeover mechanism, but they are not real solutions to address resiliency under heavy churn. Tapestry tries to address single point of failure of roots with object replication in roots, but this strategy increases control messages that lead to congestion. It also has two backup neighbors as a solution to counter failure during churn. However, the literature shows this has proven to be inadequate. Structured architectures use a deterministic search technique where the key must be known because queries are based on exact-match identifiers. It is difficult to always know the key and initiate a search.

Freenet is a hybrid structured architecture that is based on keyword search, but its root nodes are vulnerable to single point of failure. Its chain mode propagation structure makes this system unsuitable in dynamic user node populations.

Unstructured architectures work well with heavy churn, but they inherit the disadvantages of slow convergence and congestion. Here search is based on keywords that are not deterministic. Napster and Publius are not true P2P architectures, because they rely on a central server to process queries. These systems are vulnerable to single point of failure of the central servers. The partially centralized architecture employed by KaZaa and Edutella uses SNs to avoid query flooding that is common with purely decentralized architectures such as Gnutella and FreeHaven. Proper replication strategy in place for SNs may overcome single point of failure vulnerability and make them resilient under churn. Solutions are needed to address congestions near SNs, because they tend to become choke points during flash crowds. Purely decentralized architectures are not at all suitable in limited capacity networks due to heavy congestion that develops during message flooding.

TOMA generates too many control messages to maintain the multicast trees, and suffers from increased latency. It may not be supported by ISPs at most locations. Forward-when-feasible of ROMA is a good technique that avoids tightly coupled TCP connections to achieve better performance. This is possible by not having transfer rates reduced to the slowest receiver rates in the system. However, its use of traditional ACK packets is a major disadvantage, because this floods the network with packets that acknowledge receipt of messages. The other disadvantage is the use of erasure-resilient codes, because they place a toll on peers that has a negative impact on performance of the overall system. Overcast has a good tree-building mechanism, but is not resilient under churn. The concept of brokers in Narada Brokering is not a new technology, because it is similar to the concept of SNs. Its use of UDP as communication protocol is a disadvantage since it is unable to detect lost packets and there is no strategy to determine node failures.

Service-oriented e-assessment architecture is not a DE content delivery architecture. It simply provides flexible access to different assessment systems presenting one e-assessment system for use across colleges and departments in a university via integration of several assessment tools for assessing different application domains. This is therefore not a DE content delivery solution across networks.

The BT system is unique and is widely used by users who enjoy broadband network connectivity. It works well with transient nodes and has an excellent mechanism in place to address fairness. It also has the positive trait of building random graphs. Although this system appears to be suitable to use as the platform for the target architecture, many tests are needed to determine the critical combination of parameters and their metrics to meet the requirements of the target architecture. For instance, it is critical for it to work in limited capacity networks and be scalable during churn. Next, the web and tracker servers are important components of this architecture, which makes this system vulnerable to single point of failure. There must be some mechanisms in place to address this problem. The default file block size of BT is 256 KB. This is another disadvantage, especially in low capacity network environments, because its pipelining mechanism using small sub pieces of the 256 KB block would lead to heavy congestion due to exchange of excess control overheads. In succeeding chapters, the effects of changing BT to accommodate different block and fragment sizes, network and swarm size, and other parameters will be considered and simulated to determine which combinations and values optimizes the network.

2.11 Conclusion

The 18 architectures and systems presented in this chapter are primarily for music and video distribution for the entertainment industry. They are also used for software distribution. There is a need for an architecture that can enable cost-effective delivery and exchange of DE content. An extensive review of 18 popular delivery and exchange architectures and systems was presented in this chapter to conclude none of them as-is meets the requirements for efficient content delivery and exchange over limited capacity networks. This, by itself, is one of the primary contributions of this work. The other primary contribution is exploration of prior work on emerging P2P and overlay technologies to determine if any one or a few may be used alone or in combination with the target architecture. Chapter 3 presents 12 emerging technologies and describes their advantages and disadvantages.

CHAPTER 3: P2P AND OVERLAY MULTICAST TECHNOLOGIES

3.1 Introduction

A number of protocols have been developed to facilitate communications in P2P systems. However, they are not as efficient as we would want them to be. Additionally, P2P systems cannot handle many aspects of routing on their own. They need the support of the underlying network layer for packet routing in a cost-effective and timely manner. The Internet has been growing rapidly during the past two decades. Technological advances have made it possible for consumers all over the world to own computers and connect to the information superhighway. Nevertheless there are limitations to the present Internet infrastructure. We cannot overwhelm the Internet with excess data traffic or we will end up becoming victims of poor service, security and reliability of this wide area network. Since it is a long, slow process to upgrade the existing web of networks, we need to collectively use the right metrics of selected parameters of emerging technologies in critical combinations to efficiently communicate in the crowded Internet.

One potential technology for more effective use of the existing Internet is multicasting. Multicast at the network layer has been used for over a decade to address some challenges such as congestion, latency, delivery ratio and so forth being faced in the virtual information highway [77]. Where deployed, it has helped reduce traffic to some extent temporarily, but Internet multicasting has seen little deployment at the network layer. Therefore, researchers moved up to the top of the Open Systems Interconnection (OSI) stack to include the application layer in multicast routing. In doing so, they could relieve network layer routers from making routing decisions, and shift this responsibility to the overlay network layer. This is a virtual network on top of the physical network architecture [38], [78]. However, we need to keep in mind this option did not relieve the network layer from all routing activities. Rather, it somewhat reduced workload of the network layer by delegating the multicasting effort to the overlay network. Further research continued which expanded the potential for multicasting at the overlay layer. This made routing activities at the overlay layer more efficient and cost-effective [55], [56-59], [72], [74-81]. An overlay network provides logical connections between peers in a network. It does not focus on physical connections of the network. The way the overlay is built is important because queries need to be quickly resolved by peers during user searches.

There has been a good amount of work done to date in overlay multicasting by the research community, which has primarily focused on broadcasters of data as the source at the root and receivers intelligently placed in an overlay. There also has been considerable work done in the area of achieving efficient tree-based structures for these overlays. As presented in this chapter, these systems has proved to be effective in building efficient multicast trees and delivering and exchanging content in the overlay to make the system somewhat scalable, but a complete solution with these technologies is not yet available. This is because there are several complexities and difficulties of the proposed solutions, which are outlined at the end of this chapter [87].

This section is a review of prior work and current trends on 12 emerging P2P and overlay multicast protocols. The survey was made to determine whether any of them is suitable for use in critical combination with the target architecture. The answer is another primary contribution of this work.

Important categories of overlay are tree-based and mesh-based. Construction of the tree-based overlay is easier than that of the mesh-based. However, the tree breaks when a non-leaf node leaves the network, disrupting member communications in the multicast group. A simple routing algorithm is used with tree-based protocols because there is no routing loop formation during the tree construction. Communication between members in multicast groups is not disrupted with mesh-based overlays when nodes fail or depart. This is because there are redundant paths between nodes. This makes the routing algorithms more complex because they must prevent routing loops during the mesh formation. A path forwarding algorithm is used to ensure loop-free forwarding paths are constructed between group members. Mesh-based overlays also increase link stress because several duplicate messages may use a link in the forward direction.

Queries need to be resolved quickly but this must be done efficiently. Queries may be flooded to all peers in the network as is done by Gnutella described in chapter 2, but that causes significant congestion. Another approach is using random walks where peers forward queries to selected NNs at random with similar files. Yet another option is to send queries to SNs that maintain an index with metadata of files, which includes location information. However these nodes may become single points of failure and congestion points because a large number of queries is sent to them. Solutions are needed to address these drawbacks. Queries may be keyword searches or exact-match. Keyword searches are non-deterministic where the overlay is not tightly controlled and more suitable with transient nodes. But the searches take longer time and generate a large volume of packets in the network that lead to inefficient use of network capacity. Exact-match searches are deterministic and efficient, but require the user to know a key. Another disadvantage with this is its inability to resolve queries under heavy churn. With proper replication strategies, non-deterministic search works well. Replication also addresses problems associated with a single point of failure. On the negative side, replication strategies increase CPU and memory utilization of the computer and make the network more congested with unnecessary control traffic. Several replication strategies proposed by the research community were reviewed but not included in this work because of their potential complexity drawbacks.

To minimize latency, overlays may be constructed with peers belonging to the same group that are in close proximity on the physical network infrastructure. This may be measured using the ping message round-trip-time (RTT). Another proven approach is to form multicast groups with peers having similar content preferences. This enable peers within the same group to quickly resolve queries of one another.

As described in chapter 1, the condition where user nodes join and leave the network frequently is called churn, is a critical factor to consider because it is inherent in P2P networks. If the overlay cannot sustain dynamic peer population, it may cease to operate until the network structure is re-built. One of the common solutions is to have redundant load balanced nodes with plenty of capacity and resources. It is not trivial to locate peers with these features. The research community has developed and proposed a solution for locating such peers to act as backup load balanced nodes. It may be argued that this mechanism perhaps enables the overlay to be fault tolerant and scalable, but based on surveys made, it is not a practical solution because we cannot control the arrival and departure frequencies of peers. That is, we cannot enforce a policy to keep resourceful peers in the network for a longer period of time, and employ them for the purpose of load balancing. By the same token, it is important to have mechanisms in place to accurately measure churn rates. An approach is presented here to do this. This is useful information because stable peers are selected as strong candidates to be elected as backup nodes, but not quite practical because of their median session length of a few hours, which is due to their inherent property of churn.

The following sections of this chapter present all the protocols that can reasonably be considered contenders for effective P2P overlays. The Nemo [77] protocol with a solution for resiliency under churn, and FatNemo [79] protocol to address bottlenecks or choke points that are encountered at SNs. Mithos [88] and HyperCast [89] tree building protocols are described that build efficient overlay trees using simple routing techniques. Pastry [90], a decentralized, fault tolerant and scalable P2P protocol is introduced that uses efficient routing mechanism using unique node IDs. SplitStream [91] protocol is described here that uses DHT routing to improve resiliency under churn by adding links to the single multicast tree and control bottlenecks at resource intensive nodes via multiple-tree redundancy by having disjoint trees with interior nodes. The Magellan [92] protocol also employs multiple-tree redundancy to enhance performance and has mechanisms to detect node failures. Megallan-Nemo [78] is able to achieve an excellent delivery ratio. Since scalability of overlay networks is dictated by how single point of failure and fault tolerance is addressed, researchers introduced ENarada [93] protocol that attempt to do this by optimizing the structure of multicast tree using the concept of root, child and backup nodes. Swaplinks [94] peer selection algorithm is described here as a solution for peers to locate other high capacity peers for load balancing. Chunkyspread [95] as presented below uses multiple trees to balance load amongst trees to handle churn quickly, generate less overhead and scale better. The Learned model [75] discussed below efficiently creates overlay networks based on user preference. This enables queries to be resolved quickly because files with similar preferences stored in user nodes are clustered and queried first, which avoids complex search that help reduce latency and congestion over the network.

3.2 Nemo

Birrer et al. [77] introduce a P2P multicast protocol, Nemo, that achieves high delivery ratio without sacrificing end-to-end latency and costs. It uses the implicit approach [56], [91–93] for building an overlay network. Here the peers are organized in a controlled topology where data delivery method is implicitly defined per the forwarding rules. Nemo is able to withstand instabilities of the Internet by using two simple techniques. Firstly, it uses co-leaders in a cluster. Every cluster has a leader and co-leaders that are collectively known as crews. The leader of a cluster becomes a member of the immediate superior layer. Secondly, it uses triggered negative acknowledgements (NACKs). Co-leaders help improve resilience of multicast groups by minimizing dependencies on single nodes, and NACKs detect packet loss after the timeout period and request re-transmission from the neighbors storing duplicate packets in cache. This helps to control protocol overhead. The crews help improve scalability by sharing message forwarding load.

Nemo uses heartbeats that are exchanged among the cluster's peers to detect unannounced departure of peers. Once this is detected, a repair algorithm is initiated. If the peer happens to be a leader, the tree has to be repaired. Peers then elect a replacement leader from their cluster. To deal with dynamic changes, peers periodically monitor the leaders of the next higher layer. If a peer finds a leader that is closer than the current one, it switches clusters. This process also is exercised by the leaders seeking better leaders. Nemo's data forwarding mechanism has the leaders and its co-leaders forward received messages to all peers in its cluster and to the next level layer.

Birrer et al. [77] simulated different environments of Nemo to evaluate its performance with Narada [62], [68], [70], [93], Nice [61] and Nice-PRM [84]. They took into account performance and protocol overhead in their evaluation. Delivery ratio and end-to-end latency was used for gauging performance, and duplicate packet number was used to evaluate overhead. The goal was to minimize duplicate packets to improve network performance. Simulations were performed with 512 end hosts having infinite network link capacity. Two measurements of churn are used by Birrer. Under high failure rate because session time can be as low as one minute, Nemo has mean time to failure of 5 minutes and mean time to repair of 2 minutes. Low failure rate has mean time to failure of 60 minutes and mean time to repair of 10 minutes.

Simulation and PlanetLab results show Nemo fared better than the other protocols because it was able to achieve better delivery ratios with no significant delay. However, my analysis of [77] shows that there are some concerns with this protocol. First, its leader and co-leader can become choke points. Second, it is not clear what the performance will be in limited capacity wireless networks because the tests were based on wired physical infrastructure using infinite capacity. Next, there is no data on how the protocol would react to thousands of end nodes and larger payload. Finally, the packet forwarding mechanism is heavily dependent on the leader of a cluster making it vulnerable to choke points and single point of failure.

Jannoti et al. [64] say having a root node (that is similar to a SN) is a single point of failure,. Their Overcast addresses this vulnerability by having the DNS name of the root resolve to any number of replicated roots in a round robin manner. Root replication requires CPU, memory and disk utilization. It also calls for additional high capacity network links for each of the replicated roots. The solution offered by Birrer et al. [79], [99] presented in the next section addresses the root's need for high capacity network links to overcome choke points.

3.3 FatNemo

Birrer et al. [79], [99] propose fat-trees in overlays for multi-source multicast applications. The branches of fat-trees become thicker as they get closer to the root. This technique addresses bottlenecks that occur at the root of regular trees. Based on this idea, the authors introduce FatNemo, an overlay multi-source multicast protocol that is built on Nemo [77] to emulate a fat tree, where the size of cluster of peers become larger as it gets closer to the root. This protocol relies on crew members of the cluster to share the forwarded load, which help relax the burden on a single high capacity path.

Birrer points out that normal tree structures have two distinct problems. The first is lack of resiliency due to the fact the tree is dependent on reliability of non-leaf nodes. This is a challenge when there is transient node population in the tree. The next problem involves data capacity limitations where capacity is more limited as one descends down the tree. Hence, paths higher up (closer to the root) tend to become the bottleneck where response time and packet loss probability increases. The first problem was somewhat addressed by Nemo by using co-leaders and triggered NACKs. FatNemo attempts to address the second problem involving capacity limitations.

The design of FatNemo relies on three conditions. Firstly, greater capacity nodes have to be placed higher up in the tree closer to the root. Secondly, to maximize load balancing, all peers must serve as crew members. Thirdly, as one ascends the tree, size of clusters has to increase exponentially [79], [99]. Out-degree terminology is important to satisfy the first condition as this relates to the number of full-rate streams a peer is able to support. It is possible to reduce the capacity constraints of links that are higher up the tree by organizing peers based on their out-degrees [100]. The process organizes peers with large out-degree so they ascend to upper layers that help build a capacity optimized fat-tree. Large crew size is important for the second condition because it helps reduce the depth of a tree, which is important for reducing delay as it plays a critical role to control number of end-system hops. Large size clusters in layers higher up in the hierarchy help avoid root bottlenecks because this reduces number of forwarding responsibility of the root.

Birrer simulated FatNemo to compare its performance with Narada [62], [68], [70], [93], Nice [61] and Nice-PRM [84]. Performance metrics used include response time, delivered packets, delivery ratio, duplicate packets and control-related traffic parameters. They used a locally written, packet level, event based simulator called SPANS to simulate GridG [101] topologies with 5510, 6312 and 8115 nodes, and a multicast group of 256 members.

Birrer found FatNemo to be the best performer in avoiding bottlenecks in the delivery tree, as it is able to deliver most number of packets within a fixed time window during network overload. It also outperformed other protocols with response time. Delivery ratio is the ratio of subscribers that received a packet within a fixed time window. In general, delivery ratio decreases as the number of publishers increase because this causes the protocol's data delivery topology to slowly collapse [79].

FatNemo generates fewer duplicate packets per sequence number than Nice-PRM while it enjoys higher delivery ratio. This reduces the burden on the network and overhead on nodes. Control-related traffic measures overhead in the system because it measures total traffic accounting for packets at the router level. This is done by adding traffic of each forwarding router including the source node. Due to the large cluster size of FatNemo, it has more control-traffic than Nice and Nice-PRM because it selects peers with highest capacity, and does not consider distance as a metric in the selection process [79], [99]. This is a major disadvantage of this protocol in addition to the concern of not knowing how it performs in low capacity wireless network settings. Building a capacity optimized fat-tree is complex and time consuming task, especially during high failure rates, because leaders of each cluster have to often monitor other nodes in their clusters to see if there is a node with better network capacity to which to transfer leadership. Large size clusters in layers higher up in the hierarchy closer to the root can become a problem if the tree breaks during churn because it would significantly disrupt performance when a high capacity peer with large out-degree leaves the network. FatNemo also inherits the packet forwarding algorithm of Nemo, where availability of the leader or co-leader is critical because they are responsible for forwarding packets received from a peer to other peers in the cluster.

3.4 Mithos and HyperCast

Moen [102] state overlay meshes facilitate communication between nodes of the overlay. These meshes provide managed tunnels between nodes on the underlying IP network, and are established using several strategies, one of which is the use of graphical shapes like rectangles, hypercubes or Delaunay triangles [89] that have geometric routing principles and information about the underlying network.

Mithos [88] takes advantage of the geometric approach by embedding the network into a multi-dimensional space where every node is assigned a unique coordinate in this space. This method simplifies routing since knowledge of local grid coordinates is available. HyperCast [89] uses this strategy as well. This is why, once the overlay is established, there is no need of a routing protocol for the overlay. The protocol implements both the hypercube and Delaunay triangles. For Delaunay triangle, links are established per Delaunay triangulation of nodes. Here, message forwarding is done in similar manner as rectangular approach. There are two disadvantages with this approach. Firstly, it is difficult to take into account end-to-end latency to evaluate performance because the underlying network is completely ignored. Secondly, hybercube overlays need to be formed sequentially, which leads to lengthy overlay construction time in presence of a large number of nodes and churn. Mesh-based overlays are not as scalable as tree-based systems. Due to increased communication in a mesh topology, physical link stress, is increased, as are state and control overhead, and end-to-end latency.

3.5 Pastry

Rowstron and Druschel [90] evaluate the design of Pastry, a decentralized P2P object location and routing protocol that is fault tolerant, self-organizing and scalable. Each node of the Pastry network has a routing table, and neighborhood and leaf set tables with node entries that are close to the local node to minimize message travel for efficient routing. This is done using a 128 bit unique node ID for every node that is used to route a message to the active node with the ID that is closest based on latency. PAST and Scribe were built using Pastry's efficient routing properties. Simulated results show promising performance of this protocol, but the requirement of each node to maintain extensive routing table, and neighborhood and leaf set information is quite complex and resource intensive.

3.6 Learned Model

Fast et al. [75] present a method for creating overlay networks based on a learned model of user preference. The researchers use knowledge discovery techniques to create overlay networks for file sharing and compare several other methods. Common methods search files based on specific content already present in a user's library. This model is different because it has files user prefers based on the files user shares most. This eases the search process because files that are stored at a user node with similar preferences are queried first by "identifying styles" that finds nodes with files of similar interest. It is done by clustering files available in the network with hierarchical Dirichlet processes (HDPs) [75]. To create this cluster of files using HDP, it needs a list of filenames present in each user's shared library, something readily available in current P2P systems. Using this approach, an overlay network connecting users who are likely to share files with each other is created. This avoids complex search methods to reduce latency and congestion over the network, but building the overlay using this approach is not trivial.

For comparison, Fast et al. simulated four types of overlay networks. They are: 1) networks using HDP styles, 2) networks using random styles, 3) networks using direct file similarity, and 4) random networks. Performance was measured using outdegree and number of random connections parameters. Results showed after two hops, the node with total outdegree of five having all five connections to a cluster with similar interest files equivalently satisfied a query as a node having similar outdegree nodes but with three connections to a cluster with similar interest files and two random link connections. However, it satisfied more queries within one hop in comparison to the other three types of networks. The experiments also validated HDP as outperforming other approaches when number of attempted queries and network size increases [75].

The learned model in [75] is a good approach that chose HDPs to model musical styles to reach soluble queries quickly. In general HDPs tend to have the problem of putting outliers together in existing clusters rather than creating new ones. This solution does not address problems surrounding transient node population and other requirements of the target architecture.

3.7 SplitStream, Magellan and Magellan-Nemo

Birrer et al. [78] indicate better cost/benefit trade-off in networks with churn may be reached if multiple resilient techniques are collectively used. For instance, path diversity used to improve tree resiliency can be combined with a multiple-tree approach to attain better delivery ratio under churn. Multiple trees improve delivery ratios and latencies because bottlenecks are avoided by distributing forwarding load. SplitStream, Magellan and Magellan-Nemo are treated together here because they are tree-based protocols, and they adopt the multiple-tree redundancy approach.

As described earlier, protocols can be tree-first, mesh-first, DHT-first and implicit to build a control overlay for group communication and delivery tree for data forwarding. Due to their rigid structure, the latter two have been used primarily to address resiliency. Scribe [96] uses DHT-first overlay multicast protocol [90], [97] where a data delivery topology is built over a well-defined geometric structure derived from self-organizing peers, and is built upon Pastry [90] for supporting applications requiring large number of multicast groups.

Cross-link and in-tree redundancy techniques [79], [98–100] add extra links to the original, single multicast tree to improve resilience, and multiple-tree redundancy [91], [92], [106] resolves bottleneck problem by proposing several disjoint trees having interior nodes. Different stripes of data stream are forwarded over each tree that helps balance forwarding load among the peers.

The SplitStream [91] protocol takes advantage of DHT routing model properties building on Scribe or Pastry. In doing so, it is able to construct a forest with multiple multicast trees that is used to distribute data segmented in stripes over each tree to help reduce network capacity load of each node in the network, and ensure all nodes including the ones with low capacity participate and contribute by processing and forwarding the received stripe. This is done to avoid having a single point of failure and to provide fault tolerance. It is a simple and efficient method that alleviates costly network monitoring and centralized coordination. However, since it uses DHT to create the forest, trees are deep and not balanced, and this raises issues such as congestion on upstream network links and delays during periods of churn [78]. Also each node of Pastry maintain extensive routing table, and neighborhood and leaf set information that is complex and resource intensive.

Magellan [92] is an overlay multicast protocol that employ multiple-tree redundancy but builds the forest with performance centric trees [61], [104] where each peer shares its resources with at least one tree and there is a set of assigned peers in all trees for servicing requests of their interior nodes. The interior nodes consist of primary and secondary peers. Primary peers consider the tree as their primary tree. When the primary peer set runs out of resources to cater to the tree's stripe, Magellan assigns secondary peers to assist. This way, it guarantees forwarding capacity until the entire system is saturated. Magellan relies on balanced multicast trees. This helps reduce end-to-end hop distance in the distribution topology that lessens vulnerability of a tree due to node failures and minimizes overhead. It uses a per-tree maintenance protocol to detect node failure/departure and for repairing the topology, and uses lateral error recovery to recover lost messages. This allows it to recover from any of the trees and not just the forwarding one [78], [107].

Several parameters are used by Birrer et al. [78] to evaluate five resilient overlay multicast protocols. Firstly, delivery ratio is used to determine the ratio of subscribers that received a packet within a fixed time window. Secondly, delivery latency is used to calculate end-to-end delay from source to receivers. Thirdly, physical outdegree is used to see the fan out of a node for a better understanding of network capacity contribution of nodes. Fourthly, overhead is used to measure total control traffic in the system [78].

Birrer's experiments show cross-link redundancy increases delivery ratio by a negligible amount at different levels of churn. This is primarily due to duplicate packets created at some nodes when packets are randomly forwarded over cross-link, but as the failure rate increases, randomly forwarded packets help restore lost and/or missing packets. Nice-PRM also uses this technique. Nemo and Nice are tree-based protocols that use in-tree redundancy, which increases delivery ratio not sacrificing delivery latency under churn. Delivery ratio improves when lateral error recovery is combined with multiple trees. This is due to reduced packet loss. As indicated earlier, multiple trees combined with lateral error recovery allow peers to restore missing packets from other trees. However, this technique increases maintenance control traffic because it needs to maintain multiple trees. This problem may be addressed if multiple trees are combined with another resilient technique, such as in-tree redundancy that helps reduce the number of trees resulting in better delivery ratio under churn. Birrer's experiments with Magellan-Nemo show that almost perfect delivery was achieved with two trees during heavy churn. Magellan-Nice with four trees reduced average delivery latency by more than 20% as compared to Nice. Birrer concludes that overall performance may be improved by combining in-tree with multiple-tree redundancy achieving highest delivery ratio under different failure rates [78]. This may be true but makes the system complex and resource intensive.

3.8 ENarada

Xing-feng et al. [93] proposed a multicast protocol based on the Narada protocol to improve the disadvantages of tree-based overlay. This was done by combining advantages of tree-based and mesh-based overlays to construct an efficient and robust pseudo-tree overlay. It makes good use of non-leaf and leaf nodes to enhance data transmission.

The Narada protocol was developed by Carnegie Mellon University (CMU) researchers. It was one of the first application layer multicast protocols able to implement multicast functionality at the application layer. Each node has global group membership information. Every so often, it improves the mesh quality by probing and adding or dropping links that help construct multicast trees of high quality. However, it has the disadvantage that overhead becomes significant as the group size increases.

The ENarada protocol enhances data transmission making full use of each node by optimizing the structure of multicast tree instead of exploiting inner nodes of the tree. This improves usability of network resources and makes it more robust and effective. Experimental results show the protocol addresses problems surrounding single point of failure and is fault tolerant and scalable. It builds and maintains a self-organizing overlay structure by measuring network path characteristics. Here the overlay-spanning tree is constructed in two steps. Firstly, a mesh is constructed between the terminals using a Gossip protocol. Secondly, an algorithm like that of the Distance Vector Multicast Routing Protocol (DVMRP) is used to construct a reverse shortest path spanning tree from the mesh [93].

The pseudo-tree can have nodes with a maximum out-degree of six. When a new member joins the network, it queries a special node called Rendezvous Point (RP) for address of the root node. Then it queries the root node for addresses of its children node. Once it has this information, the new member requests one of the children nodes to be its father. If the child node does not agree, the new member moves on and makes the same request to one of the other children nodes and the process continues until it finds one or reaches a leaf node. The leaf node that is not an offspring may be elected as the mother node of the new member. If the father node fails, the mother node serves as backup. This ensures there is no data loss and variable delay during the recovery process [93].

Topologies in simulation experiments of [93] have 2000 routers with an average node out-degree of 3 and 6. Network topologies are created using the Transit-stub graph model generated by Georgia Tech Internetwork Topology Models (GT-ITM) topology generator. The results indicate there is no data loss from congestion. There also is no background traffic or jitter. However, when there is more than one path from the source to receiver, duplicates are received. As the overlay network structure changes, there is not much difference with link stress and average path length between Narada and ENarada. However, ENarada's overhead is high, due to exchange of control messages between multicast members and for periodic neighbor detection activity. This is a disadvantage.

Overall multicast packet loss rate decreases with ENarada and fault tolerance is better than Narada. It handles single point of failure quite well and is more scalable than Narada. This is done at the cost of creating and maintaining a mesh network. RPs are vulnerable to single point of failure. A surge in control overhead is a disadvantage when ENarada optimizes the overlay structure under churn. Also, there is no comparison data for ENarada with other protocols, except Narada, to have a better understanding of its performance improvement features in general.

3.9 Parameter-Free Peer Selection Algorithm

Steele et al. [94] argue random peer selection is common for load balancing in decentralized P2P systems. The paper looks into two concerns associated with random peer selection. Firstly, heterogeneous peer selection takes into account varying the capacities of peers, and is not parameter-free because each peer guesstimates the selection parameter since it is a prerequisite to have global knowledge of peer capacities to be able to accurately set the selection parameter. If the selection parameter is high, it leads to huge overhead for the peer. Since it is complex to determine accurate parameters, the researchers propose a method that allows parameter-free random peer selection via *sampling* technique where peers automatically compute the required parameter by approximating the global capacity view to locate peers with high capacity for load balancing. Secondly, due to lack of efficient load balanced busy peers to service requests of joining peers, the researchers propose a method using *extensions* where the walk is extended by one hop from the overloaded node, to estimate overall network utilization and determine number of attempts needed to accommodate new requests. They implement their methods over Swaplinks peer selection algorithm and discuss results of their experiments.

The Swaplinks method lets peers approximate the global capacity view to help set a more accurate selection parameter. This is done sampling the minimum capacity a peer in the network is willing to allocate to service a request. Sampling refers to sharing of capacity information by peers in the network. In doing so, over time the peers have an approximate global capacity view of the network. This effort also enables the peer to determine load status of other peers and make a decision to remove overloaded peers from its list for load balancing. As part of Swaplinks, an algorithm design is proposed and evaluated to illustrate its ability to fine-tune trade-offs between request *retries* and request *blocking*. Request retries is introduced in their algorithm when overloaded peers do not accept requests, which forces the requesting peer to find other lightly loaded peers for reply. Extensions, rather than outright retries, are used with Swaplinks, which uses fixed-length random walks as a selection mechanism. With extensions, if the node is overloaded, it extends the walk by one hop, which reduces the imposed load by an order of magnitude [94]. Results show for more than 70% of peers, the selection parameter is within 30% of ideal value indicating automatic selection parameter computations provide acceptable results [94]. The next test measured how well this method handles heavy loads. There is significant improvement when extensions are used because more than 50% of requests are denied without extensions; about 90% of the requests are accepted using extensions. QoS also was tested with favorable results. Two classes of nodes were created, where one class requires 75% assurance that requests are granted and the other class needs 25% assurance. Peers in the 75% class used extensions more than the ones in the 25% class. This raises the success rate for the peers in the higher assurance class [94]. This algorithm focuses on parameter-free random peer selection and peer capacities to enable better utilization of peers in the network via load balancing. The tests did not consider realistic churn rates and the algorithms is not a solution for single point of failure, congestion, delay, end-to-end latency, or churn.

3.10 Chunkyspread Algorithm

The swarming approach is simple and robust, but has a tradeoff between control overhead and delay. When a node receives a query that it cannot resolve, it immediately forwards it to its neighbor to minimize delay but this leads to network overhead with control messages. The tree-based approach supports continuity versus delay tradeoff, but not control overhead versus delay tradeoff. This is a problem because if a node's parent in the tree crashes or leaves, the node has to find another parent. A node can buffer the packets to keep servicing the application until a new parent is located so it depends on the tree-building algorithm to determine how much it has to buffer. The tree-based approach is more suitable for low capacity applications with stable nodes and swarming approach is better for high capacity applications with high churn rates. Tree-based can counter this by having multiple trees and sending duplicate forward error correction (FEC) codes over some of the trees that result in data overhead versus delay tradeoff.

An unstructured tree-based multicast algorithm, Chunkyspread, is presented by Venkataraman and Francis [95] that uses multiple trees to balance load amongst trees. They compare their algorithms to the DHT-based multi-tree SplitStream and show it scales better, handles churn quickly and has low overhead. Like SplitStream, Chunkyspread is a single source protocol that sends a multicast stream as *m* distinct slices where each slice is sent over a separate multicast tree. The researchers use two load parameters: 1) target load which is the volume the member prefer to send at a steady state and 2) maximum load that is the maximum volume the member may send at any time. Other parameters used by the protocol are: 1) the number of slices *M*, 2) the latency threshold, which is used to determine how the system has to weigh trade-off to achieve target load while minimizing latency, 3) minimum node degree (*MND*), and 4) minimum load (*MinL*).

Venkataraman and Francis use a default value of 16 for *M*. For example, if the target load is 100% for an application and latency threshold is 10%. 18 and 14 slices are 10% above and below 16 after rounding to the nearest slice value. So the lower edge of the range is called the *Lower Latency Threshold* (*LLT*) and the upper edge is called *Upper Latency Threshold (ULT*). Chunkyspread uses the following mechanism to reduce load balancing and latency. If node X's load is below its *LLT*, other nodes will contend to become its child to increase node X's load. If its load is greater than the *ULT*, nodes will move away from node X and become children of other parents to reduce node X's load. When the load range is within *LLT-ULT*, latency optimization effort is initiated. So it is better if the *LLT-ULT* range is large to improve latency because nodes will have less chance to get close to their target load. Chunkyspread uses the Swaplinks algorithm that generates random graph among nodes using random walks. This is used by the protocol to distribute node degrees proportionally amongst nodes with more load having higher node degree and those with less load with lower node degree. In addition, nodes are added to the neighbor set that is nearby with respect to latency to improve overall latency. Swaplinks also helps the source node discover *M* neighbors, which become the roots of *M* multicast trees and are known as slice sources. If a slice source fails, the source locates a new random node to replace the failed slice source. The performance goals of Chunkyspread are target and maximum load and latency.

The first comparison of Chunkyspread, done in [95], is on the tradeoff between *load balance* and *latency*. From the experiments, it is clear Chunkyspread performs better than SplitStream due to the load fine tuning algorithm employed by Chunkyspread. It is also evident that load in join scenario is worse than static cases with SplitStream. This is because there is not much spare capacity left with 3,750 nodes already in the network for the 1,250 nodes that are joining the network. In their experiments there are 5,000 nodes where the researchers have considered four scenarios: 1) static, where all nodes are part of the graph, 2) join, where 3,750 nodes are part of the network and 1,250 nodes join at the rate of 50 joins per second from the 20th second, 3) bursty, where a percentage of nodes fail at the same time and 4) continuous churn where nodes join and leave at the same time. The next comparison done is on *convergence time*, which is the time from start up to the last switch to a parent in the network. Two settings were used for the *LLT-ULT* range in the experiments: 1) *Lat0* where there is no latency range because *ULT=LLT=TL* and 2) *Lat2* has latency of 2 because *2(TL)/16* slices from *TL*. So if *TL=16*, then *LLT=14* and *ULT=18* after rounding up for *ULT* and down for *LLT*. It is seen from the simulations, *Lat0* converged faster than *Lat2* with Chunkyspread, and SplitStream reached a steady state when the last orphan node got a parent. Chunkyspread's algorithms with *Lat2* setting allow many nodes to have spare capacity that are able to quickly serve new nodes joining the network.

The simulations of Chunkyspread also show there is a surge in latency in the initial stage of the simulation during the load phase of the algorithm after which it slowly drops during the latency phase of the algorithm. Therefore it is important to have ample application buffering to handle the brief period of latency surge that occurs during the load phase of the algorithm. *Control overhead* is evaluated next where it is seen that peak messages are sent during times when nodes join the network and look for a parent, and while the source node starts the multicast session. Overall, there is modest overhead with Chunkyspread even during high joins. During bursty failures *Lat2* recovered faster than *Lat0* because *Lat2* has fewer hops. With redundant slices in place, the recovery time improves quite significantly. SplitStream has a poor recovery rate when 50% of the nodes fail at the same time. With continuous churn taking place for the first 1,000 seconds where there are 10 joins per second with minimum duration of 90 seconds and mean of 300 seconds, it is seen from the simulated results that the recovery rate can be improved by increasing the application playback buffer size to a modest level [95].

It is true that Chunkyspread performs better than SplitStream because it has lower control overhead, is more resilient under churn and levies less load on the nodes that in turn lead to less latency, but this is a comparison within their category because both are tree-based. There is no data on how this tree building protocol performs with a swarming-based approach using several capacity intensive applications in several churn scenarios. It also does not consider fairness in its experiments.

3.11 Conclusion

The following points of Piatek et al. [87] highlight the problems associated with tree-based approaches:

a. Poor upload capacity utilization of peers with no children.

b. Resource availability is not efficient due to improper distribution when there are peers with asymmetric download and upload link capacities because download demand exceeds upload capacity. This gets worse during churn, and calls for a solution to stabilize resource in the overlay.

c. Not robust to churn because the tree breaks when a parent leaves the network causing service interruptions.

d. Several solutions as described in this chapter may be added to address the challenges, but it is not always practical to do so because of design complexities. A simple protocol is needed to address these problems to work in heterogeneous and evolving environments.

Table 5 lists P2P architectures, systems and overlay multicast protocols that evolved over the years for building overlay networks and resolve queries. Each has its own strengths to build efficient overlay networks, address churn and single point of failure, control latency and end-to-end delay, handle congestion and link stress, and be fault tolerant and scalable. Overlay network topology and routing mechanisms of P2P systems have significant impact on these factors. It is very difficult to conclude which method is the best option because none offers solutions to all problems in meeting the requirements for design of the target architecture. It has to be done collectively. The contribution of this chapter has been the study of emerging technologies to identify and describe the merits of each and conclude that none may be used alone to provide a solution for delivering and exchanging content over limited capacity networks.

The primary contribution of this work is the selection of BitTorrent (BT) from the exhaustive list of protocols and systems for the target architecture, and determining the critical combination of parameters and their values to use collectively for designing a solution that meets the minimum requirements of the target architecture. BT has proven to be effective with bulk file transfer using its swarming feature to distribute data, and achieve better performance and resource utilization. This is because the program breaks a large file into small blocks that are distributed independently among the peers in the swarm. Since the blocks are distributed out-of-order, the peers are able to redistribute them as soon as they are received without having to wait for the entire file [87]. It is also able to handle fairness among peers in the network.

The next chapter focuses on one of the five requirements of the proposed architecture, comparing popular distance education tools to determine which one would be the best candidate to use with the selected architecture to deliver and exchange DE content over limited capacity networks.

Table 5: Summary of P2P systems and overlay multicast protocols

Protocol	Topology	Capabilities	Advantages	Disadvantages
Narada [62]	Mesh first	Not definite	Can construct high quality trees.	Significant overhead as group size increases.
Overcast [64]	Distribution tree with single root	Single source multicast	Reasonable network stress, fast convergence during node failure and use of DNS to address root single point of failure vulnerability.	Requires large number of replicated backup root nodes in case the root fails. This involves more resources and overhead.
Nice [61]	Source specific distributed tree using hierarchical control topology	P2P	Fast tree convergence and failure recovery, better link stress than Narada when group size increases (above 64) and lower control overhead when group size is greater than 32.	Does not have mechanism to address single point of failure.
Nemo [77]	Same as Nice	P2P	Better delivery ratio than Narada and Nice with the use of co-leaders; good with resiliency.	Does not address bottlenecks at the root.
FatNemo [79]	Same a Nemo	P2P	Addresses bottlenecks at the root by	Higher number of control traffic than

			increasing the size of clusters of peers as they get closer to the root.	Nice.
Mithos [88]	Geometric approach	P2P	Simplifies routing as it allows routing to be enabled with knowledge of local grid coordinates.	Difficult to use end-to-end latency for gauging performance since the underlying network is ignored.
Pastry [90]	Geometric-mesh	P2P	It builds a robust self-organizing overlay network. Scalable as it can adapt to node failures.	Does not address single point of failure.
Chord [45]	Ring	P2P	Scalable and is able to adapt to transient nodes. It is able to lookup contents efficiently.	Overlay is tightly controlled and exact match lookup is a disadvantage.
Napster [38]	Hybrid decentralized as it uses a central index server	P2P	Lookup of content in distributed peers via central index server.	Not scalable and single point of failure due to central index server.
Gnutella [40]	Mesh and purely decentralized due to flooding of requests to all nodes in the network	P2P	It is able to do fast lookup without central index server.	Not scalable since it floods queries, which consume network capacity and peer resources.
Kazaa [42]	Mesh due to partially centralized architecture	P2P	No single point of failure issue as it uses supernodes instead of central index servers. Is able to perform in the presence of highly transient nodes.	Not scalable to large networks.
Tapestry [54]	Mesh	P2P	Fault tolerant and resilient. Addresses single point of failure of root node via replication of root nodes.	Resource intensive due to replication of root nodes.
Freenet [55]	Cooperative distributed file system	P2P	Provides anonymity.	Resource intensive due to replication of files in numerous nodes. High latency because queries travel node to node.
Content Addressable Network (CAN) [52]	Mesh	Indexing system for P2P and large scale content	Scalable since the nodes are completely distributed working at the application layer and maintain small	Efficient and scalable routing and indexing is critical for good performance. Lacks security features to

		manageme nt systems	amount of state information that are able to work around node failures. This allows it to avoid a single point of failure.	resist against denial of service attacks.
OceanStore [59]	Mesh because it uses Plaxton mesh algorithm for the creation of the overlay	P2P	It is fault tolerant and scalable since content is encrypted and replicated in several server nodes. Can resist denial of service attacks.	Significant CPU and memory utilization due to encryption and large number of replicated nodes.
Scribe [96]	Uses DHT-first overlay multicast protocol to build tree above P2P Pastry network	P2P	It has positive characteristics of Pastry. Scales to large number of groups of multicast sources. Relies on Pastry for route optimization.	Fails to deliver messages if TCP connections between nodes in the multicast breaks.
Bayeux [98]	Mesh-built on top of Tapestry	P2P	Can support large multicast groups.	New joins are sent to the root, which is responsible for group management that makes the root of the tree a bottleneck and vulnerable to single point of failure. It has significant delay and physical link stress.
SplitStream [91]	Tree based with either Scribe, Pastry, Tapestry or CAN	P2P	Can support large multicast groups and avoid single point of failure. Multiple multicast trees are used to distribute forwarding load by striping data to all participants to reduce network capacity demands of each peer.	May call for memory and CPU utilization of nodes, and congestion on upstream network links during churn in the network due to deep unbalanced trees.
Magellan [92]	Tree based. Relies on Nemo for resilience during churn.	P2P	Builds forest of multicast trees using performance-centric balanced trees for data forwarding to distribute load evenly amongst participating nodes unlike SplitStream's DHT based unbalanced deep trees. It reduces	Too many maintenance control traffic. System is complex and resource intensive.

			end-to-end hop distance in the distribution topology, which lessens vulnerability of tree due to node failures and minimizes overhead.	
Overlay Aggregated Multicast Protocol (OLAMP) [60]	Tree based	P2P	Multiple groups share one delivery tree. Efficiently manages Multicast Service Overlay Network of Two-tier Overlay Multicast Architecture (TOMA). It helps reduce control overhead in the network and forwarding state in proxy nodes.	Its architecture, TOMA requires infrastructure support to enjoy the benefits.
ENarada [93]	Pseudo-tree-combination of tree and mesh	Tree formation for P2P nodes	Optimizes the structure of the multicast tree by making full use of each node for data processing and forwarding. It addresses the problem of single point of failure, is fault tolerant and scalable.	CPU and memory utilization of nodes due to the need of keeping information of nodes higher up in the hierarchy. Surge in control traffic during churn. RPs are vulnerable to single point of failure.
BitTorrent [46]	Dynamic/Random	P2P	Discourages free riding; uses random graphs suitable during churn. Breaks large files into smaller blocks.	Uses a tracker server to locate peers. Selection mechanism biased in heterogeneous networks.

CHAPTER 4: LEARNING MANAGAMENT SYSTEMS AND SYNCHRONOUS DISTANCE EDUCATION TOOLS

4.1 Introduction

DE is an effective mode of learning, if the delivery and exchange of education content are facilitated properly. Web-based DE application tools used by students and faculty are key components required to achieve this, by making the learning process easy and effective while eliminating unnecessary difficulty. Some of these tools are difficult both to learn and use, and thus turn out to be an obstacle for faculty and students [108]. A survey of popular DE application tools being used today is presented in this chapter to compare their key features, ease-of-use and learning curve. This includes comparisons of six LMSs: two developed by education institutions, and four by commercial entities. Comparisons of five SDETs are also included here: two are commercial products and three are open source. This chapter includes one of the secondary contributions: comparison of several distance education tools and selection of most suitable DE application tool to use with the target architecture.

4.2 DE LMS and SDET Requirements

DE LMS and SDET must offer a user-friendly graphical user interface, simple navigation options, and have enhanced security features to deter unauthorized access to the system and files [109]. The system cannot be network capacity intensive, and be able to process audio and video streaming in a distributed P2P network setting. Course creation and management has to be easy, and the system must support common file types. There has to be an option to reuse course contents so instructors are able to reuse contents in other sections of the same course or during another semester with minor modifications.

The virtual environment provided by DE LMS and SDET has to appear sufficiently real to create an atmosphere where faculty and students feel they are interacting in person. This raises the learning motivation of students and teaching enthusiasm of instructors that promote class participation using the *discussion board* and *chat* room. It is therefore necessary for the LMS and SDET to have easy-to-use and effective interactive communication options such as the discussion board, chat room, email, etc. [109].

Early research suggests web users need to be provided with an effective usable environment because it drives substantial savings and achieves better performance. In academia, effective LMS and SDET need little instructor time to set up and manage the course, while improving the learning experience of students. It is important for the LMS and SDET to be not cluttered with too many appealing usage options as that can be confusing for students and the instructor. Only features that meet course objectives and are relevant to a sound learning environment for designing an effective course should be included in the LMS and SDET. Since usability is critical, the LMS and SDET must be easy to use and to learn, and must offer options that are easy to remember. Considering web usability, tools must have web pages that are easy to navigate and must display information in an organized manner so users do not have to struggle to find what they are looking for. Pedagogical usability enables users to learn effectively and retain the skills and knowledge learned; it is integrated with technology usability, which is referred to ease of use and usefulness of the technology [108]. Students do not have a high degree of pedagogical usability when technology usability is poor.

4.3 Communication Tools

Zaina et al [109] describe *chat* as a synchronous communication tool that allows students to receive immediate feedback on a subject, which helps to understand group reflection of the subject matter. The researchers view *discussion board* as an asynchronous communication tool where faculty and students post messages to share and debate ideas. Positive aspects of the discussion board are that it allows fusion among the group and lets them evaluate and think about a post before responding to it. In addition, the messages are saved and can be re-visited by the class at any time. It can be beneficial if chat conversations are also saved in the event a chat message is needed at a later time.

4.4 Grade book Component

The *grade book* is a critical component of DE. Since students do not interact with the instructor in person, it is important for the faculty to be transparent with student grades. The grade book feature in most LMSs and SDETs lets the instructor post scores of home assignments, lab work, exams, etc. that the student is able to see, and monitor progress in class. This tool has to be easy to use and easy to understand by the instructor and students [108].

4.5 Knowledge Assessment

Zaina et al. [109] say knowledge assessment is essential in DE and is possible via examinations. The LMS and SDET must offer an effective tool to the instructor for developing exams with a time window to take the exam. This tool has to offer statistics of each question that gives a snapshot of how many students answered the question correctly. It enables the instructor to have a better understanding of the areas where students are deficient, and serves as useful information for the instructor to emphasize more on deficient areas to address lacking, and make necessary changes to the course content. A controversial aspect of knowledge assessment is the ability to administer proctored examinations to online students. While some educators see this as a requirement, others do not believe it has been achieved yet in a cost-effective way.

4.6 Administrator Role

Most LMSs and SDETs have an administrator whose task is to publish or set up the course, register users and address technical problems with the system. The responsibilities of the administrator should be restricted to just these tasks, because it would be difficult and time consuming for the instructor and students to depend on the administrator with course related matters, which should be the responsibility of the instructor. It would also overwhelm the administrator with problem tickets and lead to inefficient usage of the LMS and SDET [109].

4.7 WebCT, Intralearn, AulaNet, UniverSite and Col DE tools

Zaina et al. [109] compare the following systems:

- WebCT (version 3.1): developed by University of British Columbia, Canada;
- Intralearn (version 2.5): developed by Intralearn Corporation, USA;
- AulaNet (version 1.2): developed by Catholic Pontific University, Rio de Janeiro, Brazil;
- UniverSite: developed by MHW, USA;
- Col (Online Courses) version 1.0: developed by LARC (Architecture Networks Computers Laboratory), University of Sao Paolo, Brazil

WebCT was absorbed by Blackboard (Bb) in February 2006 [110] and the comparisons of [109], although dated, are presented here as a supplement to the comparisons of Bb and other distance learning tools made by a more recent journal paper of Unal et al. [108].

Following categories are used in [109] for the survey:

- System's workplace organization
- System management
- System security
- Content management
- Interactive tools
- Statistics
- Knowledge assessment

This study shows that WebCT fares well compared to other systems surveyed. Its workplace is well organized and offers simple navigation features. Navigation within Intralearn is very simple as it groups similar tools with linked words describing their functions. It also has a wizard that helps the instructor with course registration process and navigation to avoid mistakes. UniverSite has a steep learning curve, which is not the case with other systems. WebCT stands out with respect to system management because it offers better student registration options that are handled by the designer of the course. The designer can be the instructor or teaching assistant, and has privileges to manage the course and its contents, register students, and manage exams and check statistics among other things. It relieves the system administrator from course related tasks and user dependence on the administrator. As in WebCT, administrator responsibility is limited in Aulanet, which is a plus point for control by the instructor because he or she has greater control managing the course reducing dependency on the system administrator, but also requires more work of the instructor. Intralearn on the other hand has more responsibilities for the administrator to handle tasks ranging from faculty and student registration to controlling statistics. Control access by a password is implemented in all systems [109].

All systems store content in modules. However, WebCT and Col allow the instructor to create file groups where zipped files can be uploaded. Intralearn and Aulanet does not have file compression feature, so it is necessary to upload file by file. UniverSite does not have the option to store files in modules. It has to be done manually by the faculty member in an Internet server. In WebCT, users with designer privilege can copy and delete files. They are also able to edit html files inside the system.

It is possible to reuse courses in Col and WebCT. Intralearn lets the instructor duplicate an entire course, but not just one module of the course, which would be more efficient. A whiteboard feature that facilitates exchange of files between the instructor and students is available only in WebCT. Col has a similar tool that allows the faculty to use a presentation file during a chat. It also lets the instructor save chat messages from students. The other point to consider is how many chat rooms may be enabled at the same time. This is important because it is not good practice to have several participants in the same room because the same student(s) may lead the discussion most of the time. This practice also discourages others from expressing their views. For each course, WebCT has four chat rooms and one general room [109].

Zaina et al [109] indicate there are two ways to analyze student evaluations. One is via statistics and the other by knowledge assessment. Statistics show the amount of time a student spent on a course page. All of the systems surveyed offer a statistics option. However, Col and UniverSite display the results in graphic format, which makes it easier to understand. Only the system administrator is able to view statistics in Intralearn and Aulanet. This is a problem because it involves greater administrator dependence.

Automated examination options are available with all the systems. WebCT, Intralearn and UniverSite have options for creating lacuna, multiple choice and objective type exams. Aulanet does not have the feature to automatically create exams. Instead, it lets the instructor create a text file with test questions to upload in the system. The instructor has to check the exam manually. Col has the option for multiple choice exams only. UniverSite allows the faculty to decide on the questions from a question bank of any module to include in the exam, and select the degree of difficulty of the exam. Col also allows degree of difficulty selection option [109]. Table 6 is a summary of the five DE application tools surveyed in this section.

Table 6: Summary of five distance education tools [109]

DE Tool	**Workplace Organization**	**System Manage-ment**	**System Security**	**Content Manage-ment**	**Interactive Tools**	**Statistics**	**Knowledge Manageme-nt**
WebCT	Very good-excellent navigation and groups similar	Simple and very good	Yes but not strong	Allows content to be stored in	Has whiteboard Has 4 chat rooms	Yes by instructor via presenta-	Automated options for creating lacuna,

	tools			modules and creation of zipped file groups for uploads. Entire course or single module duplication possible.		tion format	multiple choice and objective type exams
Intra-learn	Very good-excellent navigation and groups similar tools	Difficult and average	Yes and strong	Allows content to be stored in modules; does not allow file compression so file by file upload required. Entire course duplication not possible; single module duplication possible.	No whiteboard Single chat room	Yes by administ-rator via presentation format	Automated options for creating lacuna, multiple choice and objective type exams
AulaNet	Good-excellent navigation but does not group similar tools	Moderate and good	Yes and average	Allows content to be stored in modules; does not allow file compression so file-by-file upload required.	No whiteboard Single chat room; only system administrator can view statistics.	Yes by administ-rator via presentation format	Instructor has to create the exam and upload text file
Univer-Site	Average-average	Difficult and	Yes and strong	Does not allow	No whiteboard	Yes by instructor	Automated options for

	navigation and some similar tools are grouped together	average		content to be stored on modules; instructor has to do this manually on the server.	Single chat room.	via presenta-tion and graphics format	creating lacuna, multiple choice and objective type exams
Col	Average-average navigation and some similar tools are grouped together	Moderate and good	Yes and strong	Allows content to be stored in modules and creation of zipped file. Entire course or single module duplicati on possible.	No whiteboard Single chat room.	Yes by instructor via presenta-tion and graphics format	Automated options for creating lacuna, multiple choice and objective type exams

Table 6 results show WebCT as the most favorable LMS. Blackboard Corporation acquired WebCT in February 2006 and phased out the WebCT name. The next section compares functionalities of Bb and Moodle LMS to determine which one is better.

4.8 Blackboard and Moodle

Unal et al [108] conducted a study on usability of Bb and Moodle LMSs. 135 students participated during fall 2008 and spring 2009 semesters. Their study show Moodle, an open source LMS, was favored by participants over commercial LMS, Bb. Only the Discussion Board module of Bb fared slightly better than Moodle. Apart from quantitative comparison of participants' responses, the authors analyzed components of both LMSs that students found useful or better than the other.

Blackboard Corporation, founded in 1997, developed Bb. It has thousands of deployments in over 60 countries and is available in 8 major languages [108]. Martin Dougiamas founded Moodle in 2001 and has over 70,000 active deployments in 222 countries translated into 75 languages [111]. The Moodle LMS is open source software and has a flexible modular design that allows users to select and implement extensions from thousands of available options to design their customized version of Moodle [108].

Unal et al. [108] asked participants about their experiences with Moodle and Bb, and provide feedback on the following components:

- Course format and layout of both LMS
- Announcements of Bb and News Forum of Moodle
- Course Documents of Bb and Lessons of Moodle
- Assignment Manager of Bb and Assignment/Activity of Moodle
- Discussion Board of Bb and Discussion Forum of Moodle
- Collaboration Tools of both LMS
- Communications of Bb and Moodle
- My Grade of both LMS

Course format and layout of Bb is quite different from Moodle because it has a layout for the instructor that is in standard compartmentalized format and cannot be changed. It has different sections for each tool that has options for the instructor to manage the course, users, and course contents. Moodle on the other hand takes a different approach because it offers the instructor to select from one of the three different formats: weekly, topics and social. Weekly format has activities organized week by week, topics is similar to weekly where each week is referred to a topic. Social format is used as the social forum. There are three columns in the default layout of Moodle. There is one broad column in the middle with two narrow side columns. For this experiment, weekly format was used with course material in the broad column. The study found students favored course format and layout of Moodle over Bb.

Announcements of Bb and *News Forum* of Moodle are the most used modules. In Bb, announcements section is on the homepage where posts can be made and seen. In Moodle, News Forum is used for general announcements and is located at the top of the center section. The participants preferred using the News Forum of Moodle over the Announcements module of Bb.

The LMS component that is used to deliver course content is critical for an online course. Bb has *Course Documents* module to provide course material in text, image and video formats. In Moodle, the *Lessons* module is used for this purpose where a lesson has a series of interactive pages. The student must select an answer in order to proceed to the next page. Students preferred the Lessons module of Moodle over the Course Documents module of Bb.

Discussion is an important component of LMS. *Discussion Board* of Bb is composed of forums where students are able to select a discussion board by clicking on the name of the board to add new topics or post a reply. Moodle’s Discussion Board creates a discussion thread automatically when an instructor creates the forum. Students are able to reply to the thread and other postings. Students rated the discussion board of Bb and Moodle about equal.

Assignment Submission feature allows students to upload assignments to the LMS. In this area, assignment is posted by the instructor with submission link for students to submit assignments by the due date. The survey revealed students favored Moodle over Bb for assignment submission.

Collaboration and group work is important for an online class to be effective. Bb and Moodle have similar elements to allow the instructor to create groups and assign students to individual group manually so they are able to share documents and send emails to each other, groups or the entire class. A Wiki module is available with both LMSs surveyed and was used by students to work together on a document, and track changes made to the document. Moodle offers an additional feature over Bb, which is the option to post profile pictures. This automatically places the student's profile picture where his/her name appears. This feature created an environment for students to get to know each other in the online environment because the profile picture linked to a profile page with description, location and email address of the student. The study found students preferred the collaboration and communications tools of Moodle over Bb.

Grade book module of a LMS or SDET is important since it allows instructors to post, update or remove grades of all students in addition to the option to import or export the grade book to an external application. The student is able to view his/her own grade using this module. Bb and Moodle have similar grade book functions providing categorization and statistical reports. The cited comparison found students favoring the grade book of Moodle over that of Bb.

This survey clearly shows Moodle to be as effective as Bb that can be used as an alternative for online courses. Moodle also offers better technology usability, leading to a greater level of pedagogical usability, and has low total cost of ownership since it is an open source LMS that does not call for licensing expenses of commercial systems. Table 7 outlines the comparison of Moodle and Bb.

Table 7: Comparison of Bb and Moodle LMS [108]

DE Tool	**Format & Layout**	**Announ-cement**	**Course Docs**	**Assign-ment Mgr**	**Discussion**	**Collaboration**	**Comm**	**Grade-book**
Bb	Not favored	Not favored	Not favored	Not favored	Favored	Not favored	Not favored	Not favored
Moodle	Favored	Favored	Favored	Favored	Favored	Favored	Favored	Favored

4.9 Synchronous DE: Network EducationWare (NEW)

Snow et al. [16] state engineering and technology sectors are dominated by classroom lecture presentation based instructions. This includes lectures by an instructor in the classroom using blackboard and an overhead projector for presentation slides. The smart classroom concept has become popular in recent years because it allows computer generated lecture presentation to be combined with annotations for display to the student audience either in the classroom or to a remote location via the Internet. Using this approach, pre-recorded lectures are used for asynchronous delivery and live classes are made possible for synchronous DE delivery and exchange. It is critical to deliver quality DE material and lectures via the Internet in order to achieve an effective synchronous learning experience. Good quality synchronous DE delivery and exchange can be reached if the students are able to receive spoken and graphical content without significant delay, are able to ask and respond to questions, and are able to interact with each other during the class period.

NEW is an open source SDET that was developed by computer scientists of the Center of Excellence in Command, Control, Communications, Computing and Intelligence (C4I Center) at George Mason University. It is able to support both synchronous and asynchronous modes of quality DE content delivery. NEW is beneficial to students and instructors that is not network capacity intensive in delivering high quality presentation without video because it is able to do so over 56 kb/s connections. This is made possible because instructor audio is compressed and streamed at 20 kb/s with quality of service (QoS) to guarantee audio delivery with higher priority. Since NEW limits individual page size to 64 KB to ensure low delay, it automatically converts larger slide pages to JPEG images to adhere to this size restriction. It does not require expensive or special hardware platform and complex software, and is easy-to-use and administer. In addition to these positive traits, the application software is entirely open source that allows users to use the source code for education purposes, and freely distribute and use the code in educational and governmental settings. The server side uses MySQL database. Apache web server with PHP scripting language is used by NEW's web portal that provides access to users for DE content. NEW is able to deliver audiographics materials composed of lecture presentation slides, annotations made on the slides, and presenter's voice to the end clients with a few seconds delay. Without video, approximately 5 MB/h of NEW recordings is required for each class time [16].

Once authenticated by the web server, students use a web portal to access live classes and pre-recorded lectures and slides with NEW. The class is presented by the instructor with a NEW client running on his/her workstation. It is easy-to-use because it is not necessary to learn several controls. Key controls to master are the recorder, whiteboard and floor control. *Recorder* is in a button layout that is used to start a recording and for playbacks. *Whiteboard* looks like a computer drawing tool on which slides are presented. Instructor is able to make annotations on the slides that make the learning experience effective and interesting. Annotation graphics are not network capacity intensive because only a few hundred bytes are generated per object. However, the freehand tool generates significant network traffic since few hundred bytes are generated per written character. The NEW client is rate limited, which prevents annotations to interfere with audio. JPEG images, HTML, ASCII text and video can be displayed on the whiteboard [16].

NEW operates in client-server mode where students and instructors run the clients, and the server is responsible for setting up connections, user authentication and content delivery. NEW is inexpensive to set up and operate because the capital outlay is minimal as it needs a basic Linux with Java application server (which can run virtually on MS Windows or Mac OS X) with a 1.8 GHz Pentium III processor, 1 GB memory and a 100 Mb/s network connection to support 40 simultaneous end clients. It is quite simple to install and operate NEW server-side components since most of this is automated. It is important to note that the number and data rate of connections supported by NEW is dictated by the available network capacity. It works with the International Telecommunications Union (ITU) video standard H.323 and is able to deliver two frames of 320 x 240 pixels per second with the capacity to transmit 30 frames per second. This number depends on the network capacity. Video function can be used by end clients with 200 kb/s or greater network connectivity. Network level multicasting cannot be used due to its limited deployment over the Internet. NEW uses the open source Transport Layer Multicaster (TLM) that allows it to use TCP to connect client and server behind Network Address Translation (NAT) gateways and firewall systems [16].

OpenSSL, an open source Secure Sockets Layer (SSL) package was added to NEW to handle authentication, and content encryption to meet the security needs of users. NEW may be adapted for use by non-English speaking users since it uses Unicode to support several languages for its display components and controls. The developers have been working to expand NEW's footprint via their on-going effort to port NEW client component suite to Linux/UNIX and Mac OS X operating systems.

4.10 Moodle Integrated Synchronous Teaching/Conferencing (MIST/C)

Pullen et al. [19] have gone beyond NEW to combine asynchronous and synchronous modes to achieve more effective delivery and exchange of DE content to students. They do so by taking advantage of software integration capabilities of a high quality asynchronous DE LMS, Moodle to combine with their existing SDET (NEW) as a basis for the design and implementation of a new synchronous online teaching system called MIST/C.

Like NEW, under control of a master client MIST/C offers audio, video, whiteboard interfaces, floor control, recorder and a playback unit. Similar to NEW, it is not network capacity intensive because it is able to operate over a 56 kb/s Internet connection without video and can support video via a better network connection.

Recommendations of George Mason University's Volgenau School of Engineering DE Committee were considered on the features that are necessary in hybrid synchronous/asynchronous online teaching environment to draw upon hardware, software and functional requirements of MIST/C as outlined in Tables 8 and 9 [18] below:

Table 8: MIST/C hardware and software requirements [18]

Hardware and Software	**Requirements**
Client OS	MS Windows XP/7; Mac OS X; Linux
Browser	All major browsers
Communication	Wired and wireless Internet, and dial-up
Security	LDAP authentication
Limitations	Seats and classes unlimited
Remote hosting	Available at moderate cost
Availability	Simple download
Audio	Internet; no separate phone connection needed
Additional hardware	Mouse, keyboard, microphone and WebCam
Responsiveness	Rapid response to user inputs

Table 9: MIST/C functional requirements [18]

Customizations	**Accessible, expandable and enhanceable**
Whiteboard	Able to accept graphic files in real time
Authoring formats	PowerPoint, PDF, Keynote, OpenOffice- all participants able to annotate slides during session
Video	Common computer formats like mpg, avi, mov and camera
Recording of sessions	Automatic on server including chat and be able to render as mpeg for podcasting
Interactions	Testing, polling and hand raising, and voice and chat
Student Tracking	Login status and participation statistics
Configurable to screen	By user and application window capture
Breakout	Able to partition class into separate groups

Changes were made to NEW so that MIST/C runs not only on Windows OS platform, but also on Linux and Mac OS X platforms. *Auto reconnect* feature was added where the Master Client informs the instructor and automatically reconnects to the server during network connection failure, without disrupting face-to-face live class or the recording session. It also automatically uploads the client recording to the server at end of class if approved by the instructor. Another useful feature, *server-side recording and download*, has been made possible where class sessions are automatically recorded on the server, in addition to the client. In the event client-side recording misses a segment of the class session, server will post that missing segment from the server-side recording to Moodle for the students or download it for other use.

The MIST/C development sought to create the simplest possible user interface. Considerable changes were made to the interface in MIST/C over NEW by integrating independent window for each active component such as audio, video, whiteboard, floor control, record control, play control, and master client into a small control window on the screen with toggle buttons to manage components as needed. A second *mirrored* whiteboard window is available for students to see full-size slides on the classroom projector; this is not cluttered with components seen on the master client primary whiteboard window. This feature is a significant advancement for simulteaching, where sets of students in different locations and in the classroom with the instructor are taught simultaneously, and is not available with any other synchronous teaching system.

The MIST/C *whiteboard* is an important component that is used to display static presentation slides and dynamic annotations. NEW supported single-page PDF, JPEG and PostScript formats for the whiteboard, but now MIST/C supports multi-page PDF and crisper PNG slides, and is able to import any application running on the client machine to the whiteboard. The *floor control* now has a button for the voting interface that can be used by the instructor to post a question, and students are able to enter their vote in real time. Breakout rooms or groups may now be formed by the instructor using the *Breakout Group Manager* feature by a button on the floor control component so that students of a group may communicate only with members of that group. The instructor is able to either join a particular group to establish two-way communication with group members or maintain supervisory or oversight role to engage in one-way communication with members of all groups [18].

MIST/C has been used extensively with multiple offerings of 17 graduate level Computer Science courses and feedback of students and faculty has been positive. Pullen et al. [135, 147] have investigated scaling MIST/C to support hundreds of courses using dynamic load balancing in a cloud environment. This would schedule new MIST/C sessions to the lightest load server platform. They tested over 200 clients in a session with a prototype of a MIST/C server in the Amazon Elastic Cloud and found it to function very well with no performance degradation. This has motivated them to include dynamic load balancing feature as a standard deployment option in the next version of MIST/C.

4.11 Elluminate vs. Macromedia Breeze

Schullo et al. [112] did a survey of two popular commercial SDETs, Elluminate Live 6.5 and Breeze 5 to analyze their features for meeting technical and pedagogical needs. Features of the two systems are listed in Table 10.

Table 10: Breeze and Elluminate features and functionalities [112]

Features and Functionality	Macromedia Breeze Version 5	Elluminate Live Version 6.5
Voice chat	Yes	Yes
Text chat	Yes	Yes
Video (two-way)	Yes	Yes
Web browsing	Yes	Yes
Interactive whiteboard	Yes	Yes
PowerPoint presentation	Yes	Yes
Polling and quizzing	Yes	Yes
Multimedia presentation	Yes	Yes
Application sharing	Yes	Yes
Hand raising and feedback	Yes	Yes
Breakout rooms	No	Yes
Record and playback (voice, text and screen)	Yes	Yes
Password secured	Yes	Yes
Plugins required	Breeze	Java
Cross platform	MS Windows and Mac OS X	MS Windows, Mac OS X and Linux

The researchers drew up the following four questions to understand advantages and disadvantages of both systems using factors like usability, technical needs, instructional needs and compatibility. The questions are [112]:

- Usability: How easy was the system to use?
- Technical needs: How well did the system meet the students' and instructors' needs technically?
- Instructional needs: How did the system help instructors and students meet the educational goals they wanted/needed to accomplish in the live sessions?
- Compatibility: How would the system integrate into an existing infrastructure?

Results are listed in Table 11.

Table 11: Advantages and disadvantages of Breeze and Elluminate [112]

Factors	**Macromedia Breeze Version 5**	**Elluminate Live Version 6.5**
Usability		
Advantages	Good for Mac users; professional looking interface; pod templates allow quick interface changes but increases learning curve; PowerPoint presentations can be made JAWS friendly for the vision impaired.	Good for Windows users; has simple interface with low learning curve; hand raising and feedback for quick feedback; keyboard access for all menus and dialog boxes for the hearing and vision impaired.
Disadvantages	User menu has confusing graphics and steep learning curve due to the complex interface.	Uploaded slides converted to low resolution graphics.
Technical Needs		
Advantages	Multiple two-way video feeds possible; good resolution of uploaded slides; full duplex audio; Flash based so limited wait time.	Functions well without video and advanced features over 56 kb/s dial up connection; auto reconnect feature; caches content if connection is slow or lost.
Disadvantages	Loses audio often and does not cache content.	Interface is cluttered with component and display icons/pods making the whiteboard appear in a smaller footprint of the window; administrative interface is complex; PowerPoint slide resolution is not clear enough to read fine details.
Instructional Needs		
Advantages	Multiple video windows for enhanced social presence; wide	Breakout rooms for groups; instructor control of all student

	variety of options available on pod infrastructure.	features; easy-to-use hand raising and emotions tools; polling can be done on-the-fly.
Disadvantages	Limited instructor control of what students do; polling requires pre-planning; no breakout rooms.	Limited use of video windows.
Compatibility		
Advantages	Supports both PC and Mac; uses common plugins.	Supports both PC and Mac; administrative back-end runs on multiple platforms, including Linux.
Disadvantages	Does not function well in a Windows environment as it does in a Mac environment.	USB microphones are sometimes difficult to setup on PC.

From this survey, it appears both Breeze and Elluminate have their own sets of advantages and disadvantages. It is therefore up to the institution to decide which SDET best meets their requirements. Since these SDETs are commercial products, their cost of ownership is quite high in comparison to an open source SDET. Next section, reports the results of comparison made in 2010 with the commercial product, Elluminate and an open source SDET, Dimdim.

4.12 Elluminate vs. Dimdim

Lavolette et al. [113] review Elluminate version 9.0 and Dimdim version 4.5, and present results in this paper. It is important to note, Dimdim is an open source SDET. Elluminate was acquired by Bb in July 2010 [114] and renamed to Blackboard Collaborate. The researchers collect participant data based on their experience with the interface and features of both systems. Table 12 lists the features of Elluminate and Dimdim.

Table 12: Features of Elluminate 9.0 and Dimdim 4.5 [113]

Features	**Elluminate 9.0**	**Dimdim 4.5**
Communications Tools		
Participants	Unlimited	20 or less
Voice chat	6 or less	4 or less
Text chat	Yes	Yes
Video	6 or less	1
Content Tools		
Guided web browsing	Yes	Yes
Interactive whiteboard	Yes	Yes
Slide presentation	Yes	Yes
Polling and quizzing	Yes	No
Multimedia presentation	Yes	No
Application sharing	Yes	No
Desktop sharing	Yes	Yes (plugin required)
Simple feedback	Yes	Yes
Logistics Tools		
Breakout rooms	Yes	No
Recording and playback	Yes	Yes
Password secured	Yes	Yes
Cross platform	Yes	Yes
Plugins required	Java	Flash

The survey had 12 Elluminate participants and five Dimdim participants attend a one hour workshop using Google applications with Elluminate and Dimdim. After the session, they were provided with a set of questions for feedback.

The researchers prepared the following five questions for the participants to determine advantages and disadvantages of each system [113]:

- Would you consider using Elluminate/Dimdim in your teaching?
- How easy or difficult was Elluminate/Dimdim to use?
- What was difficult about using Elluminate/Dimdim?
- What do you like about Elluminate/Dimdim?
- Do you have any other comments about Elluminate/Dimdim?

Table 13 is a summary of participant response to the question areas for evaluating Elluminate and Dimdim.

Table 13: Participant feedback on Elluminate and Dimdim [113]

Question Area	**Elluminate**	**Reason**	**Dimdim**	**Reason**
Would you use this tool?	Yes – 11 No – 1	Yes – useful tool; learner needs are facilitated; good supplement to a face-to-face course; multiple interactivity modes and ability to record for later viewing. No – lack of budget.	Yes – 3 No – 2	Yes- synchronous group communication; ease-of-use and no cost. No- less familiarity with Dimdim and not currently teaching.
Easy or difficult?	Easy – 8 Difficult – 4	Easy – takes short time to get used to the functions and layout; easy once you understand it. Difficult – screen shots are too small and hard to read; steep learning curve to understand microphone, camera and chat rooms; never used it before.	Easy – 3 Difficult – 2	Easy – very user friendly. Difficult – interface was somewhat difficult to use due to small screens and personal Internet connection problems.
What did you like?	Most	Joining sessions from any location; many interactive tools to use remotely and ability to record for viewing later.	Most	Very user-friendly at no cost with dynamic content and interactive environment.
Other comments?	Most	It is convenient but needs better interface design.	Most	Plan to try it.

Table 14 lists advantages and disadvantages identified by the researchers in their survey of Elluminate and Dimdim.

Table 14: Advantages and disadvantages of Elluminate and Dimdim [113]

Features	**Elluminate**	**Dimdim**
Virtual meeting room		
Advantages	Meeting room remains available if presenter logs out or network connection is disrupted.	None.
Disadvantages	None.	Closes meeting if host disconnects or logs out.
Audio		
Advantages	No audio problems encountered; has a wizard to set up audio.	None.
Disadvantages	None.	At times during the workshop, participants experienced audio problems; does not have audio wizard.
Whiteboard		
Advantages	None.	Has thumbnail of slides next to the presentation space making it easier for the presenter to navigate slides; presentation slides have good resolution; presenter's mouse pointer automatically appear as laser pointer to other participants when the cursor is in the whiteboard space.
Disadvantages	Does not have thumbnails making it difficult to navigate slides; poor resolution of images; presenter must hold down mouse button to make it appear as laser pointer to other participants in whiteboard space.	None.
Document Upload		
Advantages	Offers choice of resolution when uploading slides; moderators can upload presentation.	Each presentation is uploaded as a separate document.
Disadvantages	Number of simultaneous moderators to upload document is unlimited; adds uploaded slides to the continuous list making it difficult to find start of the presentation that was just uploaded.	Does not offer choice of resolution, just one set resolution; only designated presenter by the host may upload the document.
Text-chat		
Advantages	None.	Only the intended recipient sees the private chat.
Disadvantages	Private chat is shared with intended recipient and all moderators; private chat appears in the same window as	Private chat box blocks part of the whiteboard and cannot be removed; main text-chat window closes when

	public chat and difficult to close.	the presentation is changed.
Mood icons		
Advantages	Easy to locate and use because they are clickable buttons on the interface.	None.
Disadvantages	None.	Difficult to find and use because they are hidden under multiple layers of menus.

From the results of [113], it is evident that both systems have positive and negative traits and it is entirely up to the user to make the final selection. It is however clear that the participants lean more toward Dimdim since it is free because it is an open source SDET. But since Elluminate has some additional features over Dimdim and because it is a popular commercial product that has been in the market since 2001 and is now incorporated into Blackboard, it is being used by many educational institutions whose participants feel comfortable using it. Dimdim was launched in 2007 and has limited coverage.

4.13 BigBlueButton

BigBlueButton (BBB) is an open source synchronous conferencing tool for DE. The project started in 2007 at Carleton University in Ontario, Canada and has evolved to offer core features like chat, video, audio and desktop sharing. The BBB client runs within the Adobe Flash Player and is written in Action Script. This is scripting language developed by Adobe. The three main server components are the real-time server, application server, and voice conferencing server. These components are mostly written in Java. The server keeps all users in sync, manages and records individual sessions. The real-time server of is based on red5, an open source implementation of Adobe's Flash Media Server. The application server is a Java-based application running within Apache Tomcat, and it handles the API calls and requests of the client.

The voice conferencing server is built on FreeSWITCH, an open source telephony platform. Incoming audio packets from the BBB client are routed to FreeSWITCH through red5phone. This is an open source voice over IP (VoIP) phone. BBB does not bundle any built-in web applications. Instead, it provides HTTP based API that can be integrated with 3rd party application like Moodle [115].

The need for Flash Player is a significant disadvantage because it is unable to work in limited capacity networks. This is apparent since the minimum upstream data bit rate requirement at the user side is 500 kb/s and 1 Mb/s for download [116]. Although it offers most of the features of its open source competitor MIST/C, it does not fare well with MIST/C largely due to its network capacity intensive characteristics. MIST/C has the primary advantage of operating over 56 kb/s network capacity and other advantages such as the breakout room and mirrored whiteboard that are not available with BBB.

4.14 Conclusion

MIST/C and Elluminate have many identical features and fare well in the user community. MIST/C appears to be the most cost-effective, easy-to-use and simple distance education tool available in the market today because it is open source and has a second mirrored whiteboard for simulteaching that is not available with any other system. The comparisons of this chapter validate its rich features and functionalities, which was critical in selecting MIST/C for use by individuals with the selected architecture for delivery of DE content to remote users over limited capacity networks.

CHAPTER 5: THE CONTENT DELIVERY PROBLEM

5.1 Introduction

This chapter presents problems with the traditional data communications model to enable a cost-effective, fault tolerant, efficient and scalable solution. The problems are primarily due to issues associated with the following three major areas: 1) client-server architecture; 2) IP multicast and 3) communications media. Drawbacks of each area are described in detail to justify why they are not suitable to enable a solution for the target architecture. This analysis and justification is one of the secondary contributions of this work. The chapter is divided into three sections: 1) section one describes the client-server model and problems associated with it surrounding single point of failure, congestion and scalability, 2) section two covers IP multicast, why it is not scalable and other drawbacks limiting its growth in the Internet, and 3) section three discusses wired and wireless media such as PSTN and Enhanced Data for GSM Evolution (EDGE) factoring in their slow data bit rate in developing nations.

5.2 Client-Server Model

A server has specific services it provides to a number of clients. It serves the clients once requests from clients are received. We are able to organize data and software to run on distributed functions. This is possible because we can separate functions into clients and servers. Servers run programs that provide services to the clients. Client programs are responsible for handling user interactions and requests data or modification for the user. Multiple clients can be interacting with a single server. The term 'server' is used throughout this chapter to mean a generic server resource, whether implemented as a single server or a cluster of server machines. The server caters to the clients' requests by accessing one or multiple databases. The clients are not expected to know the details of the servers and/or databases.

Since the servers are not located on users' desks but are in secured locations, they can be securely managed. Clients are able to access several servers and enjoy the services offered by them. It is important for the network architect to consider scalability, location and security while working on the logical design of the client-server distributed architecture. The common design entails three tiers: 1) the presentation tier where client interacts with the user, 2) the processing tier where application/web server has the business logic of the application, and 3) the database tier where data is stored in database server that is accessed by the application server of the business logic layer [117]. The servers in processing and database tier are vulnerable to single point of failure and congestion, during heavy traffic volume as illustrated in Figure 9.

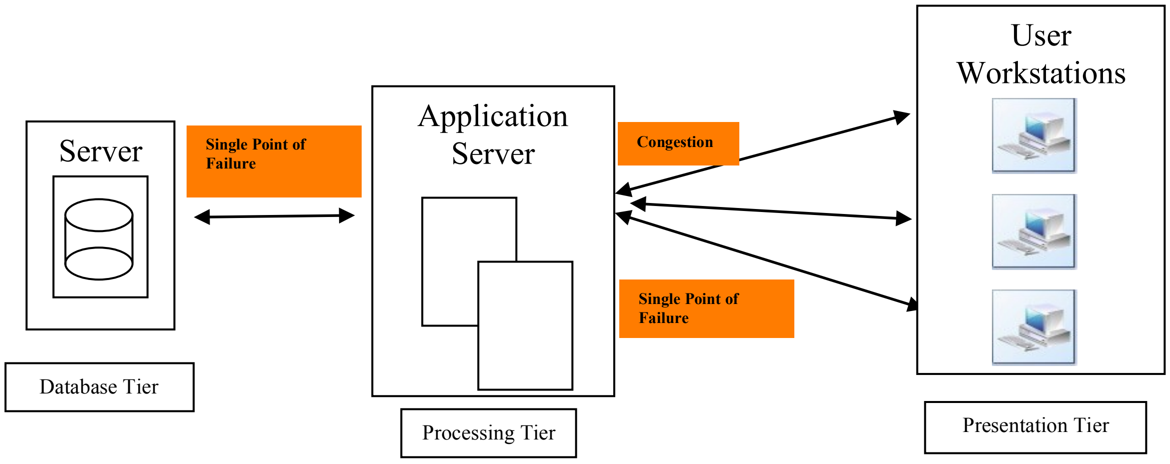

Figure 9: Three-tier client-server architecture

During the logical design, it is important to consider the distribution scheme. That is, whether it will be vertical or horizontal. In vertical distribution, different logical components are placed on several physical machines. It may be considered as multi-tiered architecture. For example, a request could come from a client to the web server of a financial institution to view web pages for online banking transactions. Once the client accesses the web page, s/he may wish to log in and check the account balance. During the log in process, the web server requests services of the application server to support this request. The application server then requests services of the database server for authentication services to verify credentials of the client in the database records. We can see from this example that a server can also take the role of a client because it is requesting services from another server. By distributing functions to individual servers, we are able to make the system more scalable and reliable because more users are able to access the system, and processing load of the servers is reduced. Whereas, horizontal distribution entail the usage of different physical machines that divides or partitions its storage area into logical equivalent parts.

Refer back to the online banking example discussed above, the web server could be replicated onto two additional servers and all three servers can be in a cluster. We would prefer to take this route to address high availability because there can be tens of thousands of clients accessing the online banking website during peak periods, and we would not want the web server to be overwhelmed and deny services to any of the valued clients. By replicating logical equivalent parts of the web server, we are able to load-balance or distribute client requests amongst three web servers as they are identical. This is a better approach but does not completely address problems surrounding single point of failure and congestion as illustrated in Figure 10.

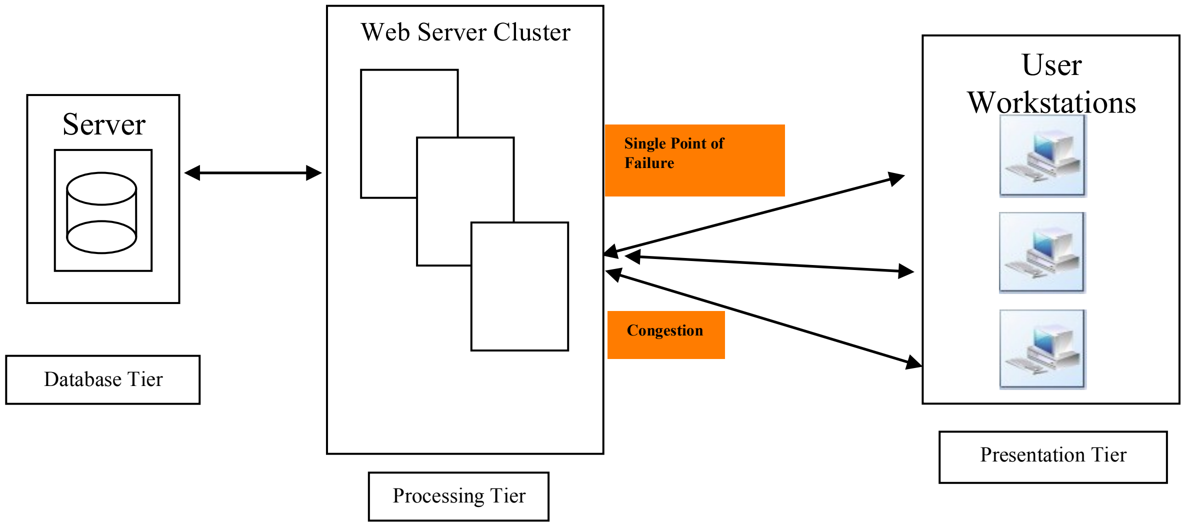

Figure 10: Load balanced web servers

5.2.1 The Client-Server Problem

Leibnitz et al. [35] point out the vulnerabilities associated with the server in a client-server architecture. The server stores applications, files or web pages to support requests of the clients and are susceptible to congestion due to heavy traffic making them bottlenecks or choke points in the network. Vulnerability associated with a single point of failure is a major concern here. If the server seizes to operate due to overwhelming traffic or malicious attacks, it will no longer be able to serve the clients requesting service. To address these drawbacks, efficient design of the network is a prerequisite to avoid such choke points. To do so, it is critical to implement load-balanced servers in the architecture so that client requests can be equally distributed amongst the servers in a cluster. It is clear from Figure 10 that a single point of failure cannot be eliminated because the server cluster may also be compromised or fail due to many reasons. On a positive note, a server storing content is secured since it is located in a trusted area, thus reducing the risk of files being distributed that have been compromised [35].

Figure 11 is notional and only represents shape of the curve, because with conventional client-server model there is an inverse relationship between number of users and data bit rate. As the number of users increase, bit rate degrades. This is also evident from the simulated results of [36], where the researchers simulate experiments to show there is significant degradation of throughput because performance deteriorates as the client population increase. Figure 12 is also notional, and shows average latency decreases with higher data capacity networks, because the congestion window of the sender grows faster. This is validated by the simulated results of Figure 13 and Figure 14. There are primarily four reasons behind this: 1) blocking and aborted transactions requiring restarts, 2) delays caused by servers' limited CPU and memory, 3) large number of queries increases network utilization and 4) limited I/O capacity. These factors prevent the client-server architecture from being a scalable solution.

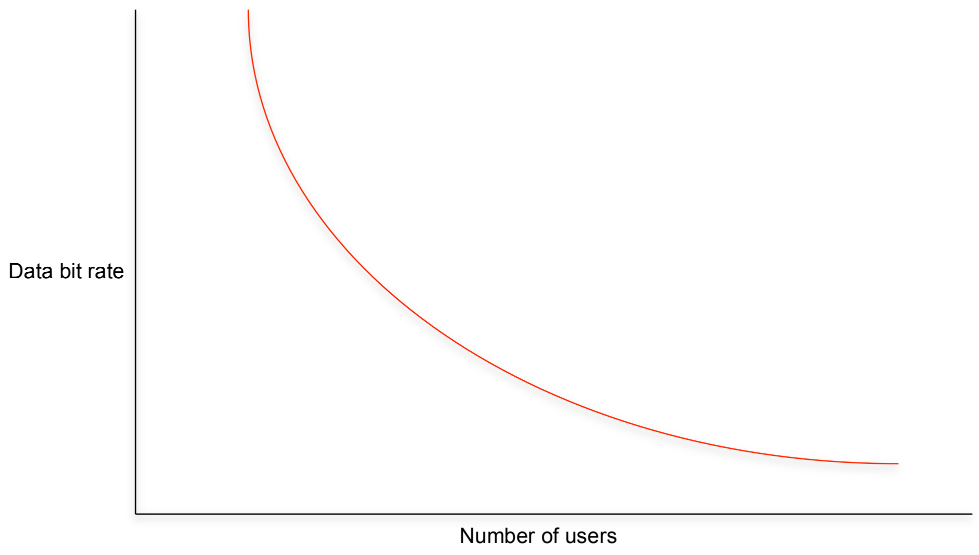

Figure 11: Notional inverse relationship between number of users and data bit rate with the client-server model

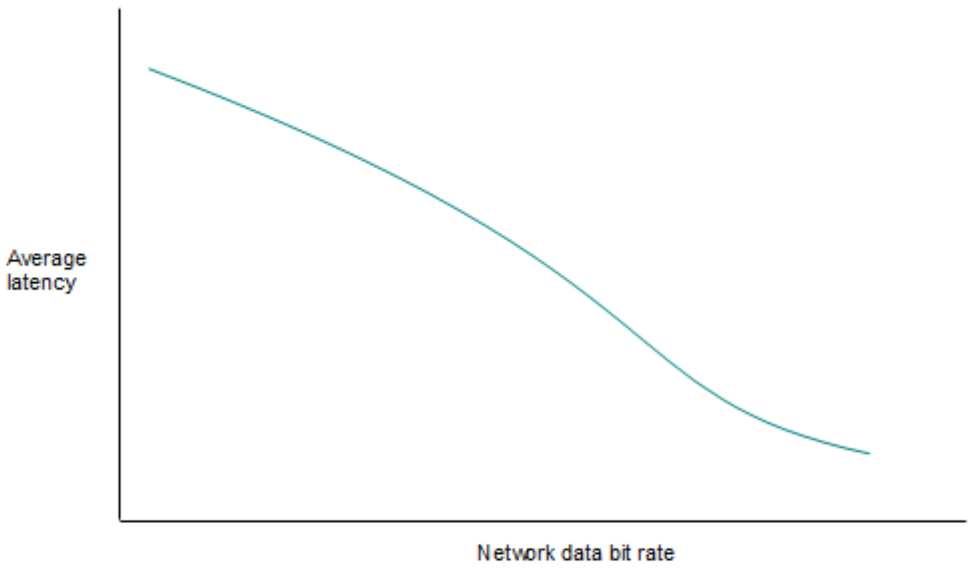

Figure 12: Notional relationship between latency and link capacity with the client-server model

This is unacceptable, which is why the pure client-server model is not an option for this work. Also, it poses the vulnerability of single points of failure because servers are centralized.

5.2.2 Tests using Client-Server Model

As proof of concept, the client-server model was simulated with ns-3; a popular and widely used discrete-event network simulator in the research community. These tests are done to validate the model is not an option for the target architecture.

To see the impact of network data rate on congestion, the client-server environment was simulated to compare the effect of sending node's TCP congestion window using the following parameters and values: 1) first set has three simulated tests with 253 nodes, 32 packets, and 10 kb/s, 100 kb/s and 1 Mb/s, respectively, and 2) second set has three simulated tests with 253 nodes, 128 packets, and 10 kb/s, 100 kb/s and 1 Mb/s, respectively. The simulated results of Figures 13 and 14 validate slow network links result in longer transmission time, because the congestion window grows slowly. This is because under the TCP protocol the slow network deters the sender from increasing the congestion window to the size of the receiver's buffer window. For example, if the receiver's buffer window is 8KB, but the sender knows bursts of more than 4 KB will clog the network, it will set the congestion window to only 4 KB. Once a connection is established, the sender sets the congestion window to the maximum segment size of that connection . In these simulated experiments, the segment size is 1 KB. The sender starts out by sending first 1 KB segment and waits for acknowledgement of that segment from the receiver before timeout. If it receives the acknowledgement, it then slides the congestion window to the right by adding the bytes of another segment, in this case, 1 KB more to total of 2 KB window. This process continues until the congestion window reaches the receiver's window size, in the event there are no timeouts. The transmission burst size is equal to the number of bytes of the congestion window. It is clear, slow network speed does not allow the congestion window to grow quickly largely because the time it takes to send and receive segments and acknowledgements, respectively, is quite long [118]. Refer to Figure 13 to see it takes over 25 seconds with a 10 kb/s network link for the congestion window to slide to the size of about 4 KB, whereas with a 100 kb/s link, the time drops to about 4 seconds for it to slide to a size of about 18 KB. With a 1 Mb/s network link, it takes only 2 seconds. Same is true for larger file sizes, and this is validated by the simulated results of Figure 14.

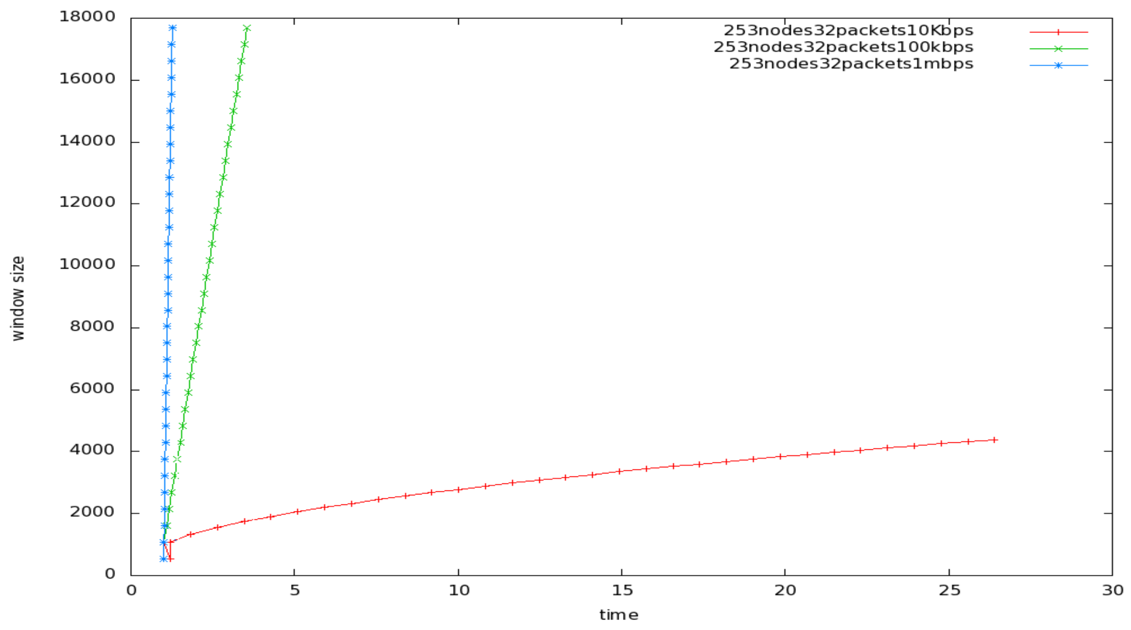

Figure 13: TCP congestion window with 32 KB file

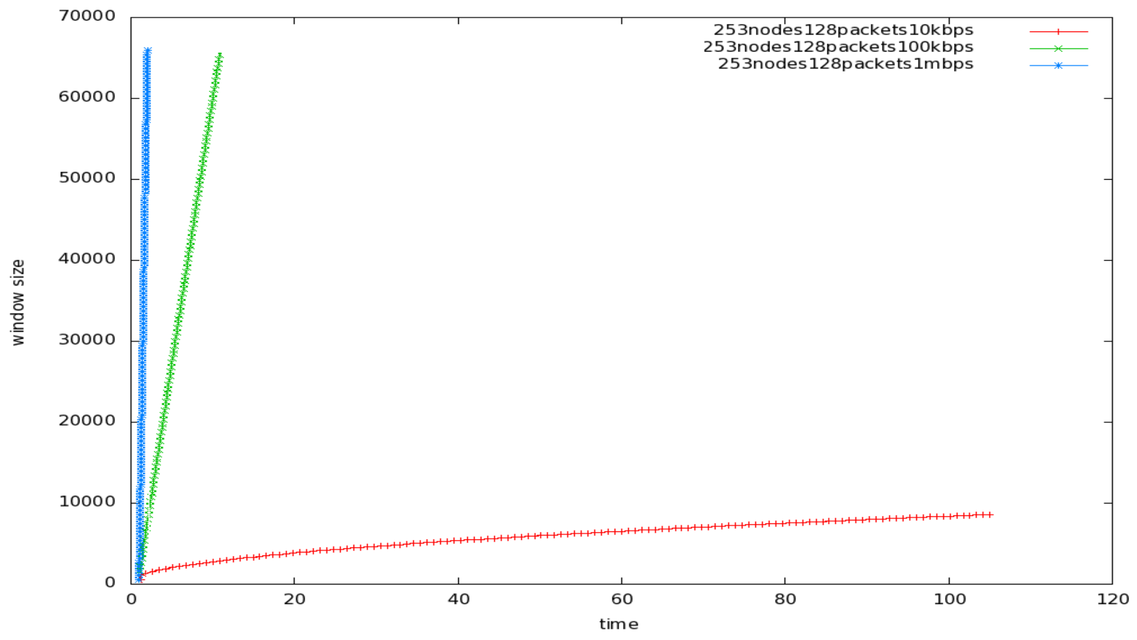

Figure 14: TCP congestion window with 128 KB file

As mentioned several times in this work, since we need to deliver and exchange content via low capacity networks, the client-server model cannot be used with the P2P Overlay Content Delivery (POCD) architecture because it does not meet this critical requirement.

5.3 The IP Multicast Problem

Unicast routing has been dominating the Internet since its inception. This has overwhelmed routers over the years due to the rise in demand of the Internet causing significant increase in data communications traffic of the user community. Network engineers had to seek alternative routing options to overcome growing congestion and offer a scalable solution. This was made possible to some extent with the introduction of IP multicast by Steve Deering in 1990 for efficient network capacity usage for multipoint communications with use of group addresses to allow network level rendezvous and service discovery [34]. Unfortunately, IP multicasting protocols did not make much headway because of deployment problems inherent in them. It is difficult to deploy and manage IP multicast, and due to its complexity, it did not make sense to deploy such a solution by the ISPs since they are not able to effectively charge for the services to their subscribers. Subscribers are not concerned whether they communicate via unicast or multicast because their primary concern is to receive good service using a suitable and efficient mechanism.

IP multicast needs network layer routers to replicate and forward packets requiring multicast-capable routers on the Internet that are not abundant as unicast-capable routers. The UDP transport protocol is used with IP multicast, which makes the solution quite unreliable since multicasting cannot adapt to congestion and recover lost transmissions. Today, most of the local networks in the Internet have multicast-capable routers. Figure 15 illustrates this is not globally true because Host A (blue) of Network C is sending a multicast message to Host B (yellow) of Network B, and Host B (green) and Host C (orange) of Network A, but Host B of Network B does not receive the message since the edge router of Network B does not support multicast as indicated by the red burst symbol. This is because the local networks are considered to be local multicast-capable domains or islands, which are connected via unicast using edge routers that do not support multicast [119].

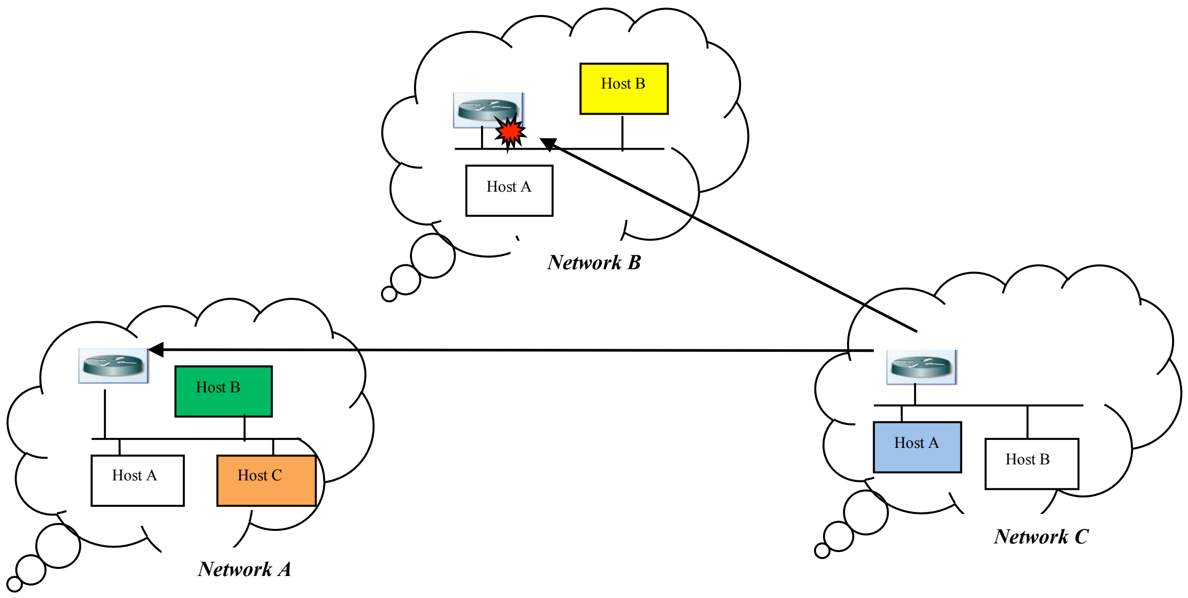

Figure 15: IP multicast problem

5.3.1 Routing Protocols

Due to lack of scalability the Distance Vector Multicast Routing Protocol (DVMRP), defined by Deering, has been limited to intra-domain routing. Using this protocol, the source forwards multicast packets to all end hosts, and the hosts receiving unwanted packets inform the source via prune messages. Multicast Open Shortest Path First (MOSPF) protocol was introduced by the IETF to build efficient trees by having a router add the list of groups for which it has local receivers to the link state advertisement packet of Open Shortest Path First (OSPF) protocol. The downside of this protocol is it can only be used in networks with routers running link-state protocols [34]. To address poor scaling of DVMRP, Core-Based Trees (CBT) and Protocol Independent Multicast Sparse Mode (PIM-SM) were developed. Here a Rendezvous Point (RP) router is introduced as the root of the single tree that is shared by all senders of that group. Packets are first sent to the RP, which then forward the packets to the receivers of that session down the multicast shared spanning tree. Scalability is improved with the shared tree approach. However, placement and discovery of the RP can be a challenge. Also, because there is huge dependence on the RP for availability of the tree, ISPs do not feel comfortable relying on each other's RPs. Multicast Source Discovery Protocol (MSDP) was developed to address this problem. This protocol eased the discovery process by having domains discover and interconnect RPs. The limited scalability problem of MSDP led to the development of Border Gateway Multicast Protocol (BGMP) for a scalable inter-domain solution that supports source rooted, shared and bi-directional shared trees [34].

Nevertheless, the complexity of RP placement and discovery has remained. Holbrook et al [65] argued this complexity can be addressed by removing RPs out of the picture. Their proposal was justified with the notion of exposing the source to the receivers so they can send *JOIN* messages to the source, which would simplify the routing process because there would be no need of RPs for this task. This solution helped make some headway of IP multicast via its adoption by the ISPs, but did not serve as a total solution since it did not address the issues surrounding the pricing model for the ISPs to charge their subscribers for using IP multicast services offered by them [34].

5.4 The Communications Media Problem

Developing nations do not have the necessary Public Switched Telephone Network (PSTN) telecommunications physical infrastructure to deliver and exchange content and audio via the data network [33]. In India, a developing country, growth of DE in higher education system between 1976 and 2001 was 20%, half of which was during 1991-2001 [120]. The slow growth of DE in developing nations is to a significant degree due to poor PSTN physical infrastructure to provide fast Internet service, which is a major obstacle for quality data communications, a critical component of DE.

This deficiency simply cannot be addressed by improving the PSTN infrastructure because it would require huge financial investments and need experienced human resources. It is a major obstacle for the growth of DE in developing nations. These countries are struggling with their finances due to their weak economies and cannot afford to revamp their existing telecommunications infrastructure to address the data communications needs of DE. Solutions are needed whereby this lack does not impede using and promoting the DE model.

High data rate wireless Internet access is not yet available in developing countries like Bangladesh. ISPs offering cable physical infrastructure Internet services do offer "broadband" Internet with data rates up to 500 to 600 kb/s. However, this service is available only to households in metropolitan areas. There are only six major cities in the country. The rest of the population lives in Districts and Upazilas. In Districts, there are several small townships called Upazilas. Districts are larger areas similar to the concept of a state in the United States. Upazilas can be viewed as a county within a state. In those areas, high speed Internet service is unthinkable. Wireless Internet service via the cellular network such as EDGE is available, but the data bit rate cannot be considered to be high speed because the maximum speed offered is about 100 kb/s.

Offering DE to an entire developing country population using existing wireless physical infrastructure without revamping it to minimum international standard is nearly impossible because the investment would be enormous. Anderson et al. [29] surveyed a group of students learning at a distant location and concluded from their research, students feel isolated when there are technological interruptions. Their findings also show low quality audio and video makes it difficult for remote students to effectively interact with the instructor and other students [29]. Low capacity Internet network connection is the root cause for both asynchronous and synchronous DE delivery and exchange interruptions.

5.5 Conclusion

It is apparent from this chapter that the client-server model is not a suitable option for the target architecture, because it is prone to single point of failure and does not have a good mechanism in place to address network congestion. Additionally, it is not scalable and is expensive to implement and maintain. IP multicast is not an option either because it is not widespread in the Internet due to several implementation and operability factors described in section 5.2. Also, IP multicast is primarily used in local networks, which is another reason to discount it from being a message forwarding and routing candidate, because the goal of this work is to have a solution in place that will enable global communication that is scalable, and be possible to implement at low cost. The design of the target architecture also has to be such that it is able to deliver content where Internet service is available via existing cable and wireless physical infrastructures. Due to limited network resources, there is an overarching need for a content delivery and exchange mechanism to operate over low capacity Internet service using cable and wireless physical infrastructure. This paves the way for chapter 6 that focuses on requirements analysis to enable a solution for the design of the target architecture and overcome these deficiencies.

CHAPTER 6: REQUIREMENTS

6.1 Introduction

Requirements of the target architecture are presented in this chapter with the intention it is able to deliver and exchange DE content and content in general to remote locations using limited network resources. The target architecture has five requirements: 1) it has to be designed, implemented and operated at low cost, 2) it has to support several languages to cater the needs of non-English speaking users, 3) it has to operate using limited capacity cable and wireless networks, 4) there may not be single point of failure nodes to enable a scalable solution, and 5) its design need to have distributed nodes to sustain instabilities of the Internet and achieve high availability. Following sections describe these five requirements in detail and justifications for their need.

6.2 Low Cost

The first requirement is *low cost* of implementation and operation of the architecture. Budget to design, implement, operate and maintain costly network architecture is not available. What is the amount that may quantify low cost? This is a difficult question to answer, but simply stated this solution has to cost an amount that can be afforded by the users. To meet all minimum requirements of the target architecture, the following six conditions have to be met at low cost with the present environment using traditional client-server model and IP multicast for content delivery and exchange:

1. Upgrade the PSTN in developing countries for high speed Internet service everywhere;
2. Provision 4G cellular service for broadband Internet service to smart devices;
3. Have multicast enabled routers on the Internet to process and forward IP multicast packets;
4. Provision redundant and load balanced resourceful clustered servers worldwide to enable fault tolerant, scalable, highly available, and non-single point of failure servers;
5. Have high data rate network connectivity to all servers to avoid congestions and bottlenecks.

Condition 1 is cost prohibitive [32], [33]; condition 2 may perhaps be considered by service providers later because it requires huge investment, so it is not timely [20]; condition 3 is impractical to do, not only because the cost would be astronomical, but also due to technical and management reasons [34] and conditions 4 and 5 are cost prohibitive [35], [36]. There is a need for other low cost and reasonable options.

6.3 Support any Language

The next requirement is *support any language*. This is important because there will be both English and non-English speaking audiences. The SDET of choice for content delivery has to use Unicode to support several languages for its display components and controls. Based on the extensive survey and comparisons of several LMSs and SDETs presented in chapter 4, MIST/C SDET meets this requirement, because it is an open source application that uses Unicode and can be modified to support several languages for its display components and controls.

6.4 Limited Capacity Network

The third requirement is to operate over *limited capacity network*. Most developing countries offer high speed broadband Internet service using cable physical infrastructure to large metropolitan cities. Small townships and rural areas rely on Internet service via an EDGE cellular network that has maximum data bit rate of about 100 kb/s. Content must be exchanged via present cable and wireless infrastructure leveraging emerging technologies without investing on alternative upgrades.

6.5 Single Point of Failure

Existence of *single point of failure* nodes is not acceptable. This is another requirement. Lecture material, audio, and perhaps video cannot be stored in and/or streamed via nodes that are vulnerable to failure and services disruption, because downtime would result in live lecture streaming interruptions, and content to be unavailable. Solutions must be in place so the architecture is always available enabling users to engage in DE sessions and exchange content from remote locations at all times.

6.6 Distributed Nodes

To meet requirement 6.5, *distributed nodes* are essential. Existence of a single point of failure may be avoided if all nodes are distributed because the nodes would not be located in a central location [35]. In a distributed setting, one node or a cluster of nodes may seize to operate without disrupting operation of the network, because there are other nodes operating independently or in clusters at different locations to take over, and provide services that were offered by the nodes that failed.

6.7 Technologies that Meet the Requirements

P2P systems satisfy both single point of failure and distributed node requirements. IP network layer for electronic packet addressing and routing does not satisfy the requirement of operating over limited capacity networks [34], [65], [119]. P2P systems are ideal for group communication on a substrate overlay network above the traditional physical IP network layer infrastructure.

As described in chapters 2 and 3, P2P systems cannot handle many aspects of routing on their own [12], [38], [45], [55], [90], [95]. They need support of the underlying network layer to handle packet routing in cost-effective and timely manner. The Internet has been growing very rapidly during the past two decades. Technological advances have made it more possible for consumers worldwide to own computers and smart devices to connect to the information superhighway [3–5]. Emerging technologies need to be in place to support the growing traffic of the Internet because existing infrastructure cannot be upgraded due to cost and resource restrictions.

To support DE content delivery and exchange, it is necessary to have powerful servers with high data rate network links to enable storage, and delivery and exchange of DE content to client nodes. This type of design calls for high speed network connectivity not only to the servers storing and furnishing DE content, but also to intermediate nodes downstream supporting end clients [35]. It would be an overstatement to say that distance education tools which operate and support end clients via low capacity network links are sufficient to achieve DE content delivery and exchange via existing limited capacity network infrastructures by claiming their features and options are capable to do this alone. This is because it is required to look at the DE content supplier nodes' network capacity as well because these devices cannot be congestion points or bottlenecks. This is why there is great need for a network architecture that can support resourceful and ordinary nodes to operate over limited capacity network environment.

The remaining three sections of this chapter discuss the set of possibilities of P2P distributed nodes and overlay multicast for the design of the target architecture.

6.7.1 P2P Distributed Nodes

DE has several components that call for a distributed P2P network architecture capable of routing packets leveraging application layer multicast. DE content storage by servers or alike is important because this has to be provided to the end clients. With asynchronous DE delivery, the content must be available to download at all times of the day and night because students and instructors (end clients) may need access to content at any time from any location. With synchronous DE delivery method, lecture content such as power point slides and other course materials must also be always available [30]. In addition, the instructor's presentation must be streamed to local instructors and students located anywhere via the Internet during the lecture period without disruption. Quality of service for voice payload must be sufficient to avoid choppy sound quality.

In addition to content storage and delivery, for both asynchronous and synchronous DE delivery and exchange, live chat sessions must be supported over limited capacity networks [11, 133].

Network performance must not degrade when there are multiple connections to the server. Availability of the server is critical for both asynchronous and synchronous DE models. The server cannot be a single point of failure. In addition, we must note that the instructor workstation or laptop cannot fail because that would then be a single point of failure. It may be possible to ensure high availability of the instructor's machine at all times with a backup workstation and/or laptop for each instructor, but that is an expensive option.

With distributed P2P nodes, it is possible to replicate DE content fragments in several peer nodes, without having to rely on one powerful server [35], [38]. In doing so, it would not be an issue if some of the nodes with replicated content fragments leave the network or fail, because other nodes with duplicate fragments would be available to provide the content to end clients needing them. This approach not only eliminates single point of failure nodes, but also lessens the need for several unicast connections to servers storing the content, which is described in the next section.

6.7.2 Overlay Multicast

The notion of multiple unicast connections is removed with the introduction of application layer multicast. By routing search requests at the application layer to a multicast group with hundreds of nodes, the need for multiple unicast connections to those hundreds of nodes in that multicast group is relieved [38], [60], [62], [102]. This is also described as multicast routing on the overlay layer. This eliminates high data rate network connections to servers, and help to meet the requirement of operating over limited capacity networks.

Unicast is used to implement overlay multicast for multicasting data on application layer. At this layer, data is replicated and distributed using computers instead of routers. As described in chapters 3, there are two types of overlay multicast protocols: 1) tree-based and 2) mesh-based [93].

The overlay tree is the most suitable structure for multicast as it offers better capacity and reduces delay. In tree-based multicast, there are no routing loops formed during tree construction, which simplifies the routing algorithm. However, tree-based overlay multicast has a negative trait. Communication between members of the multicast group is disrupted when a non-leaf member leaves the overlay. On the other hand, mesh-based overlay multicast provides redundant paths between members. In this case, if a non-leaf node leaves or fails, chances are very low for the overlay structure to be partitioned. But, there are drawbacks. Firstly, a routing algorithm has to construct loop-free forwarding paths between group members. Secondly, additional paths can be network resource intensive when data packets are sent. Tree-based overlay multicast does not have these drawbacks [93].

Overlay multicast has been successful in employing two approaches: 1) P2P networks for information sharing and messaging, and 2) overlay multicasting for group communications [102].

Since P2P systems are in an overlay above the physical IP network, its routing and location mechanisms are important considerations as they have direct impact on the system's fault tolerance, self-maintainability, adaptability to failures, performance, scalability and security [38].

P2P networks tend to have greater latency because information passes through many slow peers having long physical paths to the underlying network. Due to message flooding in P2P networks, there is congestion and inefficient use of network capacity. An overlay multicast protocol, on the other hand, is able to manage the resources of a service node and efficiently manage link stress via central control [102]. Further research has paved the way to allow multicasting at the overlay layer. This has made routing activities at the overlay layer more efficient and cost-effective [51], [77]–[80], [93].

6.7.3 P2P Distributed Nodes with Overlay Multicast Functionality

Overlay multicast using P2P systems can help in meeting the requirement of developing and operating the network architecture at low cost for several reasons. Firstly, with these technologies it is not necessary to deploy expensive routers because existing routers of the IP layer can be utilized to transmit packets over the Internet. Secondly, procurement of costly resourceful servers would not be necessary to store DE content and distribute same to end clients, because this can be done using peer nodes that are available on the Internet. Thirdly, overlay multicast significantly minimizes the need for high capacity network connectivity between end clients, and peers storing and providing content. This lets us leverage available EDGE cellular service for the target network architecture in developing nations like Bangladesh. It will not be necessary to revamp their existing PSTN infrastructure, nor, will there be the need for 3G or 4G cellular service for high speed data service to mobile devices to deliver and exchange content.

6.8 Conclusion

It is clear that the design of the target architecture is not a trivial one. There are four major critical areas to consider: 1) high availability of content, 2) quick search and delivery mechanism, 3) low implementation and maintenance cost, and 4) resiliency under churn. These have been considered and reviewed in chapters 2, 3 and 5 carefully to enable the design of a solution for the target architecture. Chapter 7 presents a framework with examples, considering these requirements, aimed toward several design options that will enable testing parameters using metrics in different critical combinations to validate performance based on simulated results that are presented in chapter 8.

CHAPTER 7: ARCHITECTURE FRAMEWORK

7.1 Introduction

P2P content delivery architectures have several advantages and disadvantages. Eighteen architectures and systems for content delivery and exchange were presented in chapter 2. Purely decentralized architectures result in flood of queries that lead to heavy network congestion [38]. Since all peers receive flooded queries, there is significant delay because the queries need to be processed by each peer and forwarded onward. Partially centralized architectures use SNs to resolve queries but these can be single points of failure [38]. They can also become bottlenecks or choke points because peers send search queries to them for location of other peers storing the content, thus a solution is needed to address this drawback. In hybrid centralized architectures, peers rely on a CS because they maintain an index of metadata and pointers to peers with content. Here, the CS is vulnerable to attacks and outages, and is also a potential single point of failure. In addition, they may be bottlenecks. These architectures employ unstructured P2P systems using non-deterministic search where peers either flood queries or rely on a CS to resolve queries thus making the server a single point of failure. Structured systems use deterministic search so that the overlay is tightly controlled. This is a disadvantage during high churn because low session peers join and leave the network frequently, and may break the overlay structure. Another problem is its need to know the key of the data file to find the node that has it because queries are based on exact match lookup [38].

Several P2P systems, overlay tree building, and multicast protocols have been developed over the years to overcome problems and facilitate content delivery. Twelve protocols to address resiliency, single point of failure, load balancing, search queries, congestion, latency, and fault tolerance were presented in chapter 3. Some of these protocols address combinations of two or three problems. None can overcome all the drawbacks. Nemo [77] addresses resiliency during churn and FatNemo [79], [99] tries to tackle congestion. Pastry [90] addresses resiliency and tries to minimize latency. SplitStream [78], [91] is built on Pastry so it also addresses resiliency in addition to single point of failure, but increases congestion and delay in the network. Magellan [92] and Magellan-Nemo [78] attempt to resolve problems with resiliency under churn. To make the system scalable, ENarada [93] offers solutions for better fault tolerance and high availability. Algorithms to quickly resolve search queries were also presented that use load balanced peers and multicast trees. The problem with these solutions is their complexity. That is, when they are included with a system, the design of that system is no longer simple. We need a simple system to use as a platform for the target architecture to address the complexities of systems surveyed.

This work aims to offer solutions to problems surrounding single point of failure, congestion, end-to-end latency, resiliency under churn, delivery ratio, search queries, scalability, and limited network capacity leveraging today's emerging technologies. It is not possible to meet this challenging goal using one default technology as-is alone, but my research shows it is possible if the right technology is selected using critical combination of parameters and metrics collectively. BT protocol fared better than the other architectures and technologies that were surveyed. It has most of the positive traits that are needed for the target architecture, but extensive simulated tests are needed to determine the best metrics for the parameters that will be used. The system addresses fairness, which is an important factor to consider in limited capacity networks. Another important point is that none of the architectures by itself accommodates the needs of asynchronous or synchronous DE delivery. They are primarily used by the entertainment industry to deliver music and video, and by the software industry for software distribution. The next section describes the framework of the target architecture. It sets the stage for the logical design of the target architecture to simulate with different parameters and metric values for achieving best results to use as the design of the selected architecture.

7.2 Content Delivery Framework

Efficient data communication and fast search querying over the Internet are not simple tasks. The objective of this chapter is to illustrate the whole communications process flow that consider all the processes. Four important areas of the content delivery framework are introduced here: 1) group formation, 2) content distribution, 3) search process and 4) presentation methodology.

The goal of this work is to provide a solution for users to quickly access content at any time from any location over the Internet using limited capacity wireless network infrastructures. The following sections present a use case that describes the steps of each of the four areas of the framework involved in exchanging distributed content. This is another secondary contribution of this work. It entails a simple example of a user joining a P2P network to search, distribute and retrieve content of a lecture presentation chapter using limited capacity networks. This example clearly illustrates each step of the four areas outlined above.

7.3 Group Formation

The first area is group formation. It involves two steps shown in Figure 16:

1. In step 1, a student or instructor, also classified as an ordinary node (ON) or peer (P), first installs the custom BT [46] client applications on his/her personal computer (PC), laptop or smart wireless device.
2. During step 2, ON contacts the standard web server (WS) to download a torrent file that has information about the file, its length, name, tracker server's URL, etc. Once the torrent file is downloaded, the ONs client application connects with the tracker server (TS). The tracker is responsible for helping peers in the swarm find each other. Upon receipt of the location information of other peers with the entire file or pieces of the entire file that the ON is looking for, it directly establishes connections with those peers.

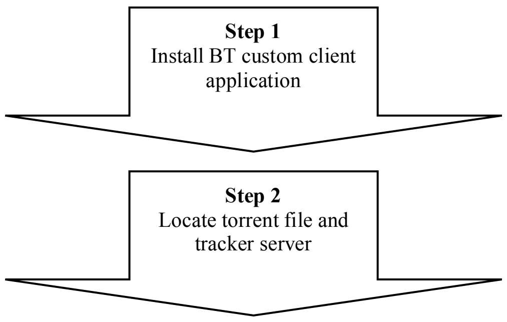
Step 1
Install BT custom client application
Step 2
Locate torrent file and tracker server

Figure 16: Group formation

7.4 Content Distribution

The next area of the framework involves DE content distribution. Figure 17 illustrates five steps:

1. In step 1, a peer with the entire file also known as a *seed* or *seeder* contacts a standard web server to upload a torrent file.
2. During step 2, content is requested by an ON. It contacts the WS for the torrent file that directs it to the respective TS with the information of the *swarm* of peers that have the file the ON is looking for. For example, lecture presentation of textbook chapter 1 has been requested by the ON. This chapter is being distributed to ten ONs in the swarm. The new ON joins the swarm and is able to download the pieces of chapter 1 from the peers that have been downloading the same amongst themselves.
3. In step 3, the ONs in the swarm share the file fragments (FFs) of chapter 1 so each has the entire presentation to view, listen, and share.
4. The ONs periodically inform their respective tracker in step 4 about the content they have in storage. This ensures data replication across ONs in the swarm to address single point of failure.
5. In step 5, the tracker updates its index as ONs join and leave the swarm. For example, if five (1-5) of the ten ONs suddenly leave the network, chapter 1 lecture would still be available for download from remaining five ONs (6-10). When ONs (1-5) ungracefully leave the swarm, the tracker detects this event via ping messages and immediately updates its index, and does not refer new requesting nodes (RNs) to the ONs that left. Index update process is similar to the process when new ONs join the network.

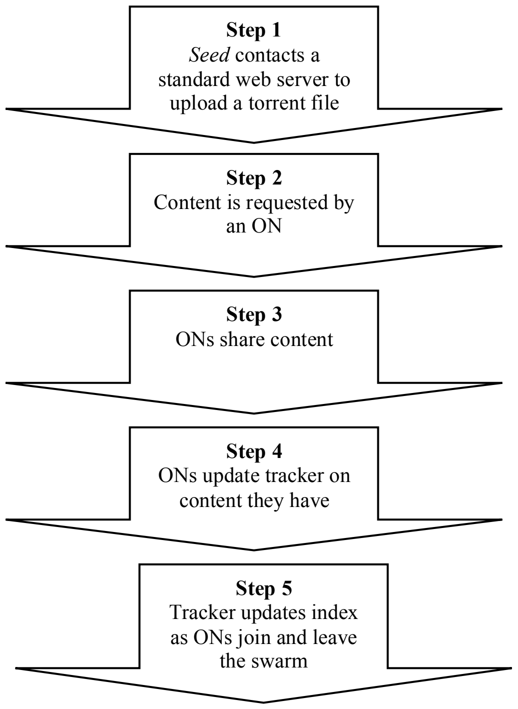
Step 1
Seed contacts a standard web server to upload a torrent file
Step 2
Content is requested by an ON
Step 3
ONs share content
Step 4
ONs update tracker on content they have
Step 5
Tracker updates index as ONs join and leave the swarm

Figure 17: Content distribution

7.5 Search Process

With reference to section 7.2 group formation:

1. Based on the torrent file it received from the WS, the ON contacts the TS to join the swarm of ONs with the chapter 1 presentation. Once the joining node (JN) joins the swarm, it is now ready, in step 1 of this area of the framework, to request chapter 1 presentation. It initiates communication with the ONs that have some FFs of the file to download.
2. In step 2, if one of the ONs that are downloading the FFs grants connection to the JN, it downloads the FFs from the ON, so JN now has the FF.
3. JN then establishes concurrent connection with four ONs in the swarm with the best

downloading capacities via a mechanism known as *unchoking* during step 3, and starts uploading the FFs it is downloading from other ONs. It is important to note the JN initially downloads the rarest pieces first in the swarm and uploads those pieces to the other ONs. This is done to quickly distribute the rarest FFs in the swarm so it is no longer a rare piece.

4. In step 4, the JN compares the downloading rates of the ONs it is connected to with other ONs in the swarm to see if there are other ONs with better downloading speed rates. If so, it *chokes* the ON it wants to disconnect with, and unchokes the new ON with better downloading rate that it wishes to include in its concurrent connected ON group of four peers. This mechanism enables all ONs to maintain connections with ONs with the best network link capacity. These steps can be seen in Figure 18.

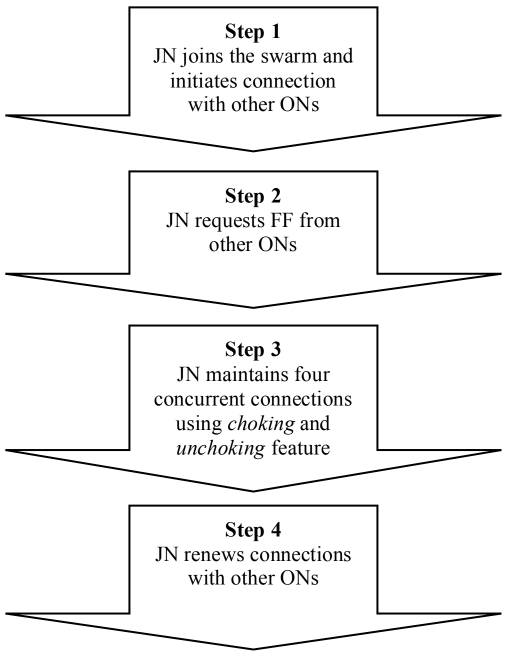
Step 1
JN joins the swarm and initiates connection with other ONs
Step 2
JN requests FF from other ONs
Step 3
JN maintains four concurrent connections using *choking* and *unchoking* feature
Step 4
JN renews connections with other ONs

Figure 18: Search process

7.6 Presentation Methodology

This area entails personalized user interface of the node. This is important because the node has to join the P2P network using a browser that is able to interact with web pages from a web server. It will also need to interact with the local SDET server to initially log on and join a class session. A standard graphical user interface (GUI) along with software programs and plug-ins are required on all nodes of the custom P2P network to be able to seamlessly communicate and interact with the servers and other nodes.

7.7 Content Delivery Architecture

Logical design of the content delivery architecture is presented in this section. Content delivery is a process. It involves several components of different layers working together for efficient content delivery. Legacy delivery methods are resource and network capacity intensive as described in chapter 5. They use a client-server model using traditional IP packet forwarding that is slow, and neither fault tolerant nor scalable. Here I introduce distributed nodes operating at an overlay network to overcome these limitations.

To have a scalable solution in place, as described in chapters 2 and 3, five design goals are important: 1) P2P distributed nodes communicating in a rich overlay network, 2) robust search mechanism in place, 3) possess efficient connection mechanism, 4) have high degree of fault tolerance and 5) be resilient under churn.

These goals can be met with five measures: 1) determine number of peers in the network factoring in their arrival and departure rate, upload and download capacity, and resource and capacity information, 2) performance does not deteriorate as size of the network grows, 3) break large files into smaller fragments for distribution, 4) group peers with similar preferences, and 5) have fairness.

Efficient overlay formation is possible by using peer proximity information of the underlying network. This is difficult because nodes join and leave the network frequently. It is therefore ideal if node capacity measurements and similar file preferences are used for this purpose. Next, nodes need to have local information instead of a global view to enable them to locate and evaluate peers already in the network to build a better overlay. The BT protocol uses swarms to group peers with similar file preferences, and uses choking and unchoking mechanisms for efficient communications between peers in a swarm.

POCD architecture is a content search and delivery framework that overcomes limitations of existing architectures to present a better platform to enable fast content search, delivery and exchange by the nodes in limited capacity networks. The core feature of this four layer architecture is distributed storage and efficient routing by the distributed user nodes.

Four layers make up the proposed content delivery architecture. They are access layer, presentation layer, build and search layer, and data layer. The architecture is a major contribution of this work and is illustrated in Figure 19. The following sections describe each of the layers.

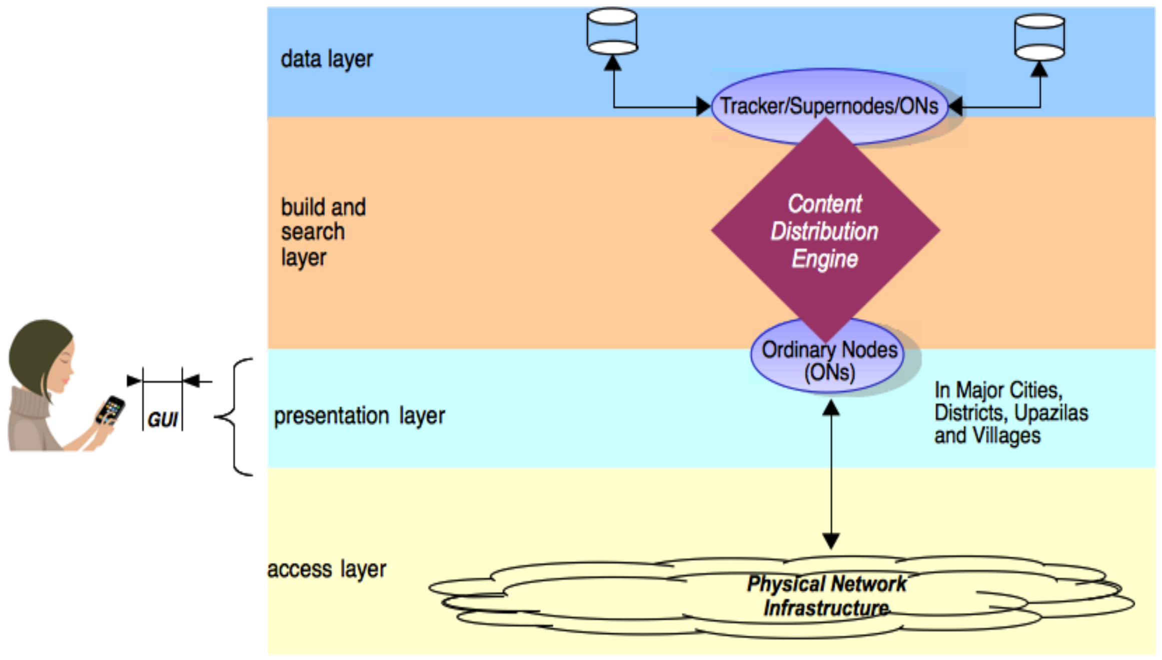
data layer
Tracker/Supernodes/ONs
build and search layer
Content Distribution Engine
Ordinary Nodes (ONs)
GUI
presentation layer
In Major Cities, Districts, Upazilas and Villages
access layer
Physical Network Infrastructure

Figure 19: P2P Overlay Content Delivery (POCD) architecture
Image of woman using smartphone [121]

7.7.1 Access Layer

The *access layer* is the underlying physical layer network infrastructure of the PSTN, Internet service providers and cellular service providers to access the Internet.

7.7.2 Presentation Layer

The *presentation layer* has the *node* or *peer* and the *search agent*. The user accesses other layers via this layer using the GUI to join a swarm and generate search queries for content, and to interact with SDET and other P2P applications.

7.7.3 Build and Search Layer

The *build* and *search layer* has the *content distribution engine* that consists of the *index engine*, and the *build* and *search agents*. This is the core layer of this architecture and a detail description with an example is presented in this section. It also includes the BT protocol for overlay formation and content search. There are *ordinary nodes (ONs)* and *tracker servers (TSs)*. TSs are more powerful machines and maintain a database with identifiers of all files the ONs are sharing. When the ON launches the BT custom client application, it establishes connection with the standard *web server (WS)*, and downloads a torrent file that has information about the file, its length, name, tracker server's URL, etc. The torrent file seamlessly connects the ON's client application to the TS that helps the ON locate the swarm or group of peers that is sharing the file it is searching. Upon receipt of the location of other peers with the pieces of the entire file that the ON is looking for, it directly establishes connections with those peers. ONs periodically update the tracker with information on the file pieces they have and their resource utilization. This information is updated in the index of the tracker. Since the large file is broken into small fragments of 32 KB and 128 KB, it makes the distribution process fast and efficient. It also scales the network better because file sharing can be achieved quickly as the number of peers grows since more peers are able to share their pieces with each other, and have the entire file faster. The choking and unchoking mechanisms described earlier are unique and facilitate efficient and effective distribution of content. It is also possible to control the number of neighbors an ON can have and the swarm size. Many experiments were done to determine the best critical combination of parameters and metrics for a solution of the target architecture, and the results are presented in chapter 8. The swarm basically has the number of ONs that are sharing pieces of the same file. This information may be kept in the cache of each ON in the network. This is useful information because it enables the requesting ON to quickly unchoke ONs with the best downloading capacities in the swarm.

The user node initiates a query using the search agent. The distribution engine receives the query and refers to the index engine for a match. The index engine has an index of metadata and pointers to nodes storing that content. This is updated in short intervals by the build agent via discovery service of the search agent. If there is no match to the target query, the search agent probes other nodes for a match and upon discovery updates the build agent. The build agent feeds this discovery to the index engine and the process continues.

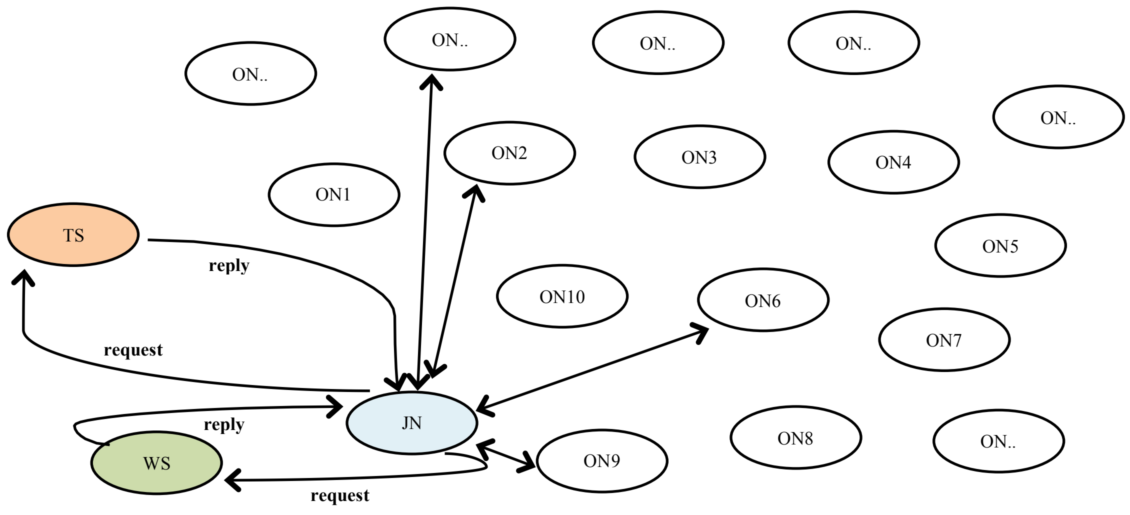

Figure 20: POCD architecture search example

Refer to Figure 20 for a simple search request example. This diagram and example is also one of the secondary contributions of this work. It was intentional to keep this asymmetric (no direct connection between ONs other than the ones that are connected). Since there may be hundreds of ONs in the network, ON numbers beyond 10 are denoted with dots (..). Here, JN in blue needs, e.g., algebra 1 chapter lecture and connects with the standard web server (WS) in green that has the torrent file associated with the sought file. Algebra is used here as an example; the technology applies equally to any subject content. The WS replies back with the torrent file, which has information about the file, its length, name, hashing information, and tracker server's URL. Once the JN receives the torrent file, its custom client application connects with the TS in orange, which provides IP addresses of the ONs in the swarm that are sharing the pieces of the file. JN then requests connection with one of the ONs in the swarm, and requests the file. Once the JN receives the first piece of the file from that ON, it then initiates four concurrent connections to the ONs in the swarm that have the best downloading link capacities. In this example, they are ON2, ON6, ON9 and ON.. . As mentioned earlier, the JN monitors the swarm every ten seconds to see if there is any other ON with a better downloading rates. If there is, it establishes connection with that ON, dropping the one it is connected to that has the poorest downloading rate. This process continues until the JN has the entire copy of the file. TSs and WSs can be *single point of failure* and ONs join and leave the network quite often. To counter single point of failure of TSs and WSs, backup servers would be ideal. TSs and WSs are not responsible for storing and sharing files so their task is limited to providing location information of the ONs of a swarm. The data bit rate requirement of the TS and WS is very low. The TS provides a random list of peers to the JN that creates a random and unstructured overlay. It does not need any special join or recovery algorithms when peers join or leave the network so control traffic is minimal [122]. Random graph is better than tree or mesh overlays because during churn power law graphs tend to get segmented [46].

Since peers do not have a global capacity view, uniform distribution of chunks in all peers is maintained using *rarest-first policy* protocol where rare chunks are downloaded first. *Endgame mode* protocol is also used so an ON with most of the chunks of a file does not have to wait for a long period to download the remaining few chunks by being able to ask all ONs for the chunks rather than waiting on slower peers with the chunks [46], [50].

7.7.4 Data Layer

The data layer simply consists of several data sources with presentation files, user data, index and routing tables, HTML and multimedia files, and images. These may be stored in database servers, ONs, TSs and WSs.

7.8 Conclusion

Overlay distributed nodes are able to satisfy queries faster than the legacy model. Availability is much higher because content is replicated in distributed nodes, contributing to fault tolerance and eliminating single point of failure. Application layer routing and communication via the peers is fast because this is done by the nodes instead of network layer routers. Indexes of TSs with peer location information facilitate the search process and enable fast self-organization in transient node population. Rarest-first policy ensures rare pieces of a file do not remain *rare* for a long period of time thus enabling faster distribution of all the pieces of the file. Choking and unchoking mechanisms described earlier facilitate optimization of network and node capacity utilization, and increase resiliency during churn.

A framework and logical design of the target architecture was presented above. To meet the requirements of the target architecture, it is important to determine which parameters are critical for optimizing the system, and how to use them collectively with the best metric values. Determining this is a major contribution. This was not trivial and required an extensive number of tests in a simulated environment. Chapter 8 presents a top-down ordered structure for the tests, and based on this structure simulations were done. Results of the tests are compared and presented to arrive at the best solution. An *n*-tier hierarchical training model is also presented in chapter 8 to show how the selected architecture may be used to train students and trainers at the bottom of the hierarchy with limited network capacity connections in remote locations.

CHAPTER 8: EXPERIMENTS TOWARD THE DESIGN OF THE SELECTED ARCHITECTURE

8.1 Introduction

This work has achieved a significant analysis and potential synthesis of DE software and overlay network components. However, a huge design space remains to be explored. This chapter describes a set of experiments, conducted via simulation, to define the "sweet spot" of effective system performance using the technologies analyzed previously.

Chapter 6 outlined requirements and chapter 7 presented a framework of the target architecture. Chapter 2 explored 18 architectures and systems but none is able to meet all requirements of the target architecture alone or as-is to enable content delivery and exchange via limited capacity networks. Chapter 3 outlined 12 emerging technologies on prior work to ascertain which ones would be suitable to use collectively to enable a solution that meets the requirements of the target architecture. As discussed in chapter 7, it is critical for the target architecture to operate in limited capacity physical layer infrastructure with dynamic distributed nodes. Chapter 5 validates with justifications and simulated results that the client-server model is not suitable for this work. This chapter will present simulation results to validate that the P2P model meets the needs of the target architecture. Due to heavy churn rates, structured P2P systems would fail because they work well in a more stable environment. Structured design has solutions in place to quickly resolve search queries since they are based on deterministic search, but it is necessary to know the key for an exact match. Also the search takes place in a *N-node* system where each node maintains information about $O(\log N)$ other nodes in its routing table, and is able to resolve queries via $O(\log N)$ messages to other nodes [38]. The advantage is that as the number of nodes increases exponentially, search time increases linearly. However, due to heavy presence of user nodes with low session times, this model degrades in performance because it defeats the purpose of exact match since the peers with key fragments are not always present in the network. This notion directs the design to unstructured design of the architecture where it is based on keyword search. It does not require the user to know the key and is able to operate well without degrading performance under churn. That is, if technologies are in place with unstructured design it is able to quickly adapt in dynamic environments with unstable peers that join and leave the network frequently [38], [77]. The problem in combining several technologies to the system makes the design complex and difficult to manage.

Other factors considered are how fast the system is able to resolve queries since there is no exact match. The overlay tree has to be built efficiently and quickly. When non-leaf nodes leave the network, the overlay has to be able to quickly re-establish the tree. As stated earlier, tree-based overlays are able to construct the tree faster than mesh-based solution but the tree breaks as non-leaf nodes depart the network. On the other hand, they do not need complex routing protocols to maintain the tree as mesh-based trees do. However, mesh-based trees generate too many control messages flooding the network causing unnecessary congestion which is a major disadvantage in limited capacity networks [93]. This is why neither a mesh-based nor a tree-based overlay tree building mechanism is suitable for the target architecture. It is better to have a random graph selection mechanism in place because as described above, the peer selection algorithms presented by the research community result in a power law graph that gets segmented during churn. Random graph selection is achieved with swarms of BT [122].

It is critical for the delivered data rate not to degrade as number of users and packets per second increase. Also, average latency including distribution latency may not increase, as this would directly impact performance. These factors are affected by issues surrounding resiliency under churn, single point of failure, network congestion, and peer overload. Keeping this in mind, many combinations of parameters and metrics were simulated to determine the best solution as presented in this chapter.

8.2 Parameters

The following parameters were taken into account for the simulations to determine the optimal parameters because they contribute to slower data bit rate and higher latency when node population and number of packets grow:

- Link stress
- End-to-end delay
- Transient nodes
- Query resolution
- Content replication
- Single point of failure
- Load balancing

This section is an overview on how the parameters listed above negatively impact performance due to data bit rate degradation and poor latency.

8.2.1 Link Stress

Link stress increases as node population grows. Simulation results of [96] indicate link stress rises as number of packets increase impacting latency due to delay. This is directly proportional to the number of users because as nodes join the network, they exchange messages with other nodes that include control fields and queries. Having peers with similar file preferences in a group or swarm helps reduce link stress because unnecessary queries to peers that do not have the file being searched can be avoided. This helps to minimize flow of control traffic in the network. Node degree is the number of virtual connections a node has with other nodes in the overlay using the physical link. Large node degree leads to greater link stress because contention for the interface is more that lead to congestion, packet loss and poor performance. This is because a node with higher node degree than its physical connections will encounter multiple copies of the same data. Latency goes up as group size becomes large, so it is important to keep the size of node degree small for user nodes with low capacity physical network link. Random overlay graphs generate minimal control traffic because there are no special join or recovery operations as peers join and leave the network. These can be achieved with BT protocol because the node degree is limited to five and the overlay graph is random and unstructured [122].

8.2.2 End-to-End Delay

The time it takes for a message to traverse from node A to node B is end-to-end (E2E) delay and we do not want this to be high. To achieve a good user experience, Voice over IP (VoIP) E2E delay has to be within 150 ms and some of the online games require it to be under 100 ms. Due to the rapid growth of Internet traffic, E2E delay has become a major concern because it negatively impacts quality of service (QoS) of delay critical applications such as voice. The researchers conduct experiments using PlanetLab testbed and show 2-Hop overlay routing algorithm (2-Hop ORA) improves round trip time (RTT) over traditional IP layer routing. This is possible if each node maintains RTT information with the other nodes in the network, which becomes impractical as the overlay size grows [123]. Several scalable overlay protocols have been evaluated in this work to determine the best solution for controlling E2E delay. BT fares well because of the use of swarms where nodes with similar file preferences are in a swarm and connect with each other to share files [46], [122].

8.2.3 Transient Nodes

Performance deteriorates as peers join and leave the network frequently [45], [50], [52], [67], [77], [78], [91], [95]. When a peer with a FF of interest leaves the network ungracefully, it is no longer available for other peers that wish to download its content. When the peer's departure is detected by the TS, it updates its index and the process is quite similar when peers join the network. This increases the number of control information in the network that lead to congestion, packet loss, delay and poor performance. Churn is a problem with tree-based overlays because non-leaf node failure or departure breaks the tree. Several protocols have been evaluated, and BT was selected for simulations because it does not use trees and has the most effective mechanism in place to minimize control information during churn.

8.2.4 Query Resolution

Performance improves when searches are resolved quickly [42], [45], [52], [75], [80]. To facilitate fast searches data has to be replicated on ONs and a TS has to maintain a metadata index to assist peers with their queries. Napster [38] uses a central index server to locate nodes storing content being searched, but this raises the concern of the server being a single point of failure. The Gnutella [124] system floods the query in the network that increases congestion. Several search techniques that are better than the options used by Napster and Gnutella have been presented by the research community and described in chapters 2 and 3. Search mechanism using torrents, tracker servers and the BT algorithm are included in the simulations of this work, and results of different parameters and metrics are presented here to determine the best options.

8.2.5 Content Replication

Replicated and clustered content greatly improve performance of the network because it is able to overcome problems surrounding churn and node failure [75], [77], [80]. There are several ways to replicate content across distributed nodes in the network. BT's replication techniques have been evaluated and simulation results are presented in this chapter.

8.2.6 Single Point of Failure

Having a single point of failure is one of the primary factors that deter scalability. This was discussed in chapter 5 where the client-server model was presented with examples describing the server being vulnerable to single point of failure [35], [36]. Distributed nodes communicating in an overlay significantly addresses single point of failure problems because it is not necessary to depend on a central source. TSs of BT system help resolve queries faster than other solutions but the TSs may also be single points of failure. During implementation of the selected architecture, the TSs need to be replicated to ensure the service is not disrupted when it goes down.

8.2.7 Load Balancing

Performance improves if peers are efficiently load balanced [78], [81] because load balancing reduces dependencies on single nodes by sharing message forwarding loads with several other nodes. Several mechanisms are in place in the research community to locate lightly loaded peers with long session times to serve as potential candidates for load balancing needs [78], [91], [96]. Scribe partitions and distributes forwarding load over multiple nodes to improve scalability because each node forwards multicast messages for a small number of groups [96]. These are attractive proposals but they are complex. After careful evaluation, my analysis determined that the choking and unchoking mechanism of BT is suitable for load balancing because peers with slow downloading rates are automatically dropped from the connected group of peers every ten seconds [122].

8.3 Data Bit Rate

Content delivery and exchange via limited capacity networks is a critical requirement of this architecture. Each end node has to be able to access distributed content and support DE classes not sacrificing its data bit rate. Achieving this gets more difficult as the number of users increase. The full potential of P2P systems can be seen with large scale deployments, which is not possible with the client-server paradigm. The *network effect* is missing in client-server deployments, but is present with P2P applications because the value of the network goes up for the user as the number of user nodes increases. This is because a larger number of nodes offers aggregated storage capacity of peers in a group facilitating file availability to the user, for which there is linear relationship between file availability and user node population, not sacrificing response time and throughput [40]. Figure 21 is notional and represents the shape of the curve showing the ideal linear relationship between node population and file availability.

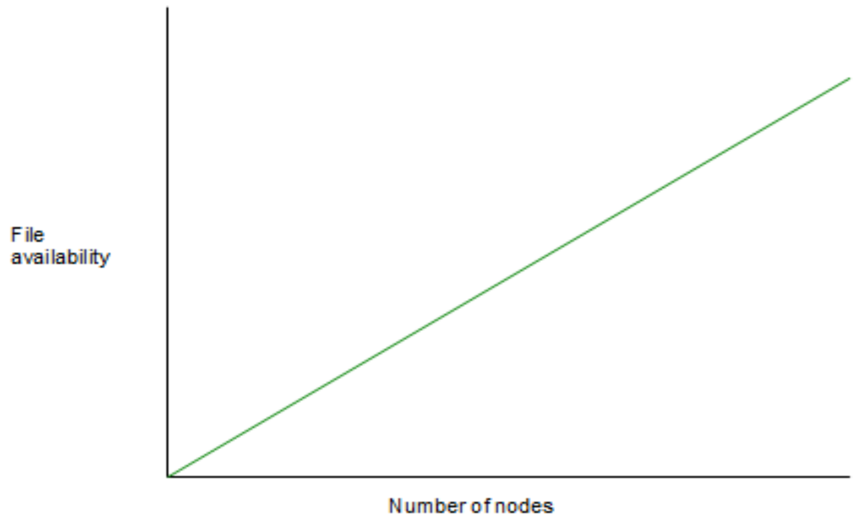

Figure 21: Notional linear relationship between user population and file availability with P2P systems

Since P2P systems are used with the target architecture, it is important to consider the impact on bit rate when the number of users increase. Figure 22 below is also notional and illustrates the convex shape of the curve of traditional P2P distributed systems.

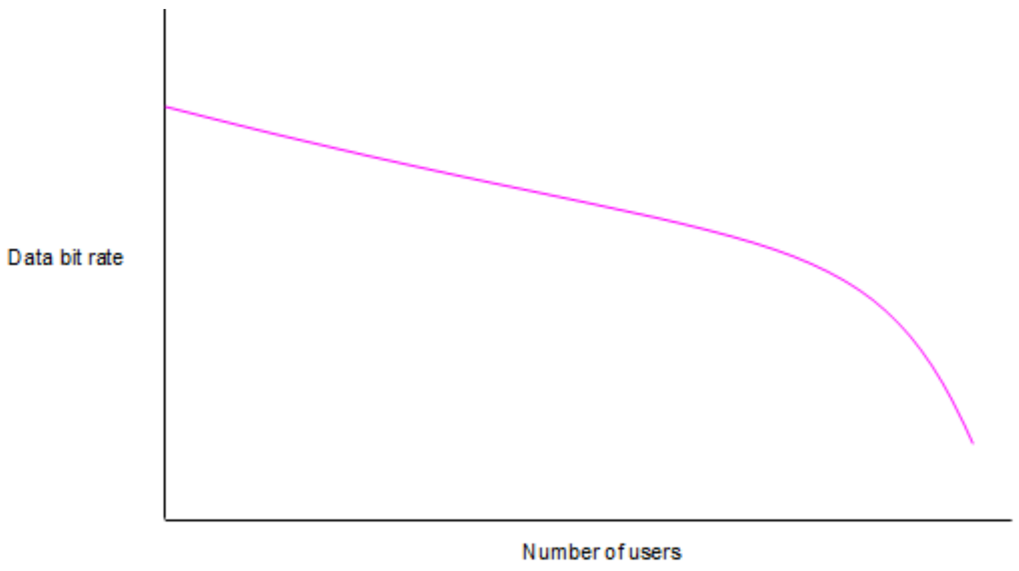

Figure 22: Notional non-linear relationship between number of users and data bit rate with P2P systems

The slope of the curve is convex showing a negative relationship between user node population and data bit rate. The curve gets steeper as the number of users grows because as user node population increases data bit rate falls, *i.e.*, they are inversely proportional. This is particularly true as nodes join and leave the network. P2P networks are prone to high degree of transient peers with low session times [45], [50], [52], [67], [78], [79], [91], [95]. The other factor for this inverse relationship is limited network capacity of the overlay tree structures having multi-source and capacity intensive applications. This is because capacity decreases as one descends into the tree [79]. This information helps our investigation to exclude tree formation, and determine the right combinations of BT parameters and fine-tune them to minimize this inverse relationship and achieve the best design option for the target architecture.

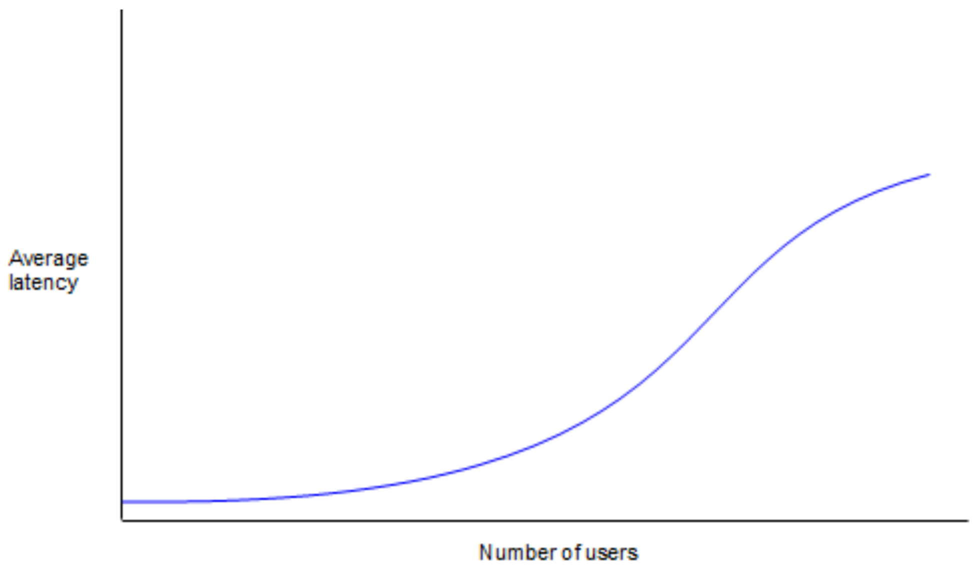

Figure 23: Notional relationship between user population and average latency with traditional P2P systems

Figure 23 notionally illustrates a convex curve where average latency increases exponentially as number of users increase, over a limited range. This is because there is contention for resources by more users, causing delays that lead to significant wait time. If some users wait one second for a resource, the tenth user will need to wait nine seconds [125]. This is not acceptable so a solution has to be in place to address problems surrounding it.

One of the primary goals of the P2P approach is to ensure performance improves as node population grows. This is because having a large number of nodes increases aggregate storage and file availability, leading to better search results, not necessarily at the expense of response time. Latency is, however, directly proportional to user population and number of packets generated per second. The simulations include these parameters to determine a solution that would not negatively affect data bit rate and average latency including distribution latency, when there is a surge in number of users and packets because they may lead to congestion, E2E delay and node failure. These problems must be controlled to enable the best solution for the architecture.

8.4 Top-Down Ordered Structure

It is important to design simulation experiments such that they achieve their goals efficiently. Five design goals outlined in chapter 7 have been taken into account in developing the top-down design structure for the simulated experiments. This section describes this top-down structure and is illustrated in Figure 24.

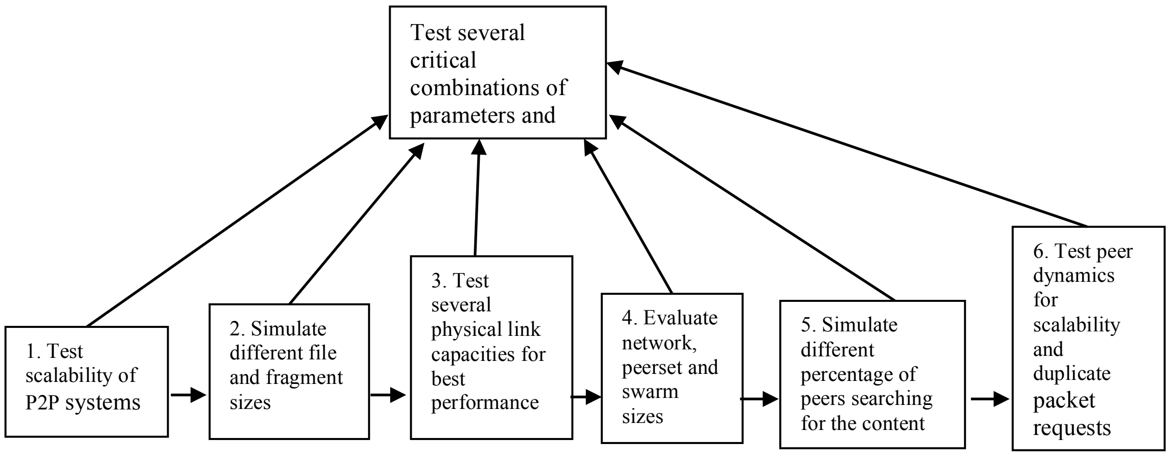
Test several critical combinations of parameters and
1. Test scalability of P2P systems
2. Simulate different file and fragment sizes
3. Test several physical link capacities for best performance
4. Evaluate network, peerset and swarm sizes
5. Simulate different percentage of peers searching for the content
6. Test peer dynamics for scalability and duplicate packet requests

Figure 24: Top-Down ordered structure of simulation experiments

This top-down structure entail, *Test several critical combinations of parameters and metrics* box at the top tier and six boxes with respective simulation methodologies in ordered progression at the second tier. Each of the second tier methodologies is simulated with different parameters and metrics to determine the best solution. The best solutions of each of the six methodologies of tier-2 are then combined and tested to achieve best results under churn in limited capacity networks. The boxes at tier-2 are numbered and described below:

1. It was demonstrated in chapter 5 that client-server model does not scale well. This was validated theoretically and via simulated results. Chapters 2 and 3 covered P2P systems where its scalability potentials were described. In this chapter, simulated result of BT P2P environment is presented to show the system is scalable.
2. It is important to know what file size may be used for the peers to quickly download and share with other peers. As stated earlier, BT breaks a large file into small fragments. These fragments are shared by the peers in the swarm to receive the complete file in the shortest possible time. The goal here is to determine the suitable file and fragment size to achieve best performance. These parameters are simulated with different metrics along with constant values of other parameters. The other parameters used here are network size, link speed, swarm size, peerset size, and maximum growth. Peer dynamics is set to zero and duplicate requests is set to one for these simulations.
3. The primary goal of this work is to have a solution in place where content can be delivered and exchanged via limited capacity networks. Using the parameters of item 2, link capacity parameter is evaluated with different metric values to determine if the system can be optimized in scenarios where peers have low capacity physical links.

4. Once the suitable file and fragment sizes, and physical link capacity is determined, simulated results of different network, peerset and swarm sizes are evaluated and presented. This is done to see which combination of these parameters with best metrics should be selected as the optimized solution for the target architecture.
5. In this step, different number of peers of the network size is simulated to determine the values that achieve fastest distribution of the entire file amongst all the peers searching for the file.
6. It is a fact that peers join and leave the network quite often (churn), and disrupt the network. It is important that the network be resilient during high churn rates. This step entails simulations with peer dynamics parameter to see whether the system is resilient when large number of peers join and leave the network.

The results of the parameters and metrics described in numbers 1 to 6 of tier-2 are compared independently to determine the best solution to overcome the drawbacks. The best solution of each test methodology is selected and collectively combined, and further tested with the parameters of the next steps and so forth, until the final set of parameters and metrics achieves the optimized solution. This critical combination is selected for the design of the selected architecture to deliver and exchange content over limited capacity networks.

8.5 Scalability of P2P System

This section presents the simulation results of the popular P2P system BT in an environment with several parameters using different metrics. This was done with a reputable and leading P2P simulation tool, PeerSim. This is widely used by researchers globally [126]. The parameters of the simulator are described next [127].

File size: This is the total size of the entire file that the peers are trying to download and share.

Fragment size: The large file is broken into blocks of 256 KB and then each block is further broken into 16 KB fragments.

Maximum swarm size: This parameter specifies the dimension of the cache of each node. That is, information about the number of neighbor nodes that are sharing the same file held by each node.

Peerset size: This is the number of neighbor nodes contained in the *peerset* message that is sent by the TS to a node that joins the network, to let them know the set of nodes the newly joined node may communicate with to download the file blocks of interest.

Duplicate requests: These are the number of duplicate *request* messages that can be sent to different peers after an *unchoke* for the same block is received. For example, if the value of duplicate requests is 4, a node may request every block to 4 different neighbors.

Maximum growth: This value determines how much the network can grow when *Control Network Dynamics* is used. While the *network size* parameter specifies the total number of nodes in the network at the start time of the simulation, but this value defines to what extent the network may grow. For example, if the size of the network is 50 and the *maximum growth* value is set to 30, then the network may scale its dimension up to 80 nodes.

Simulation results shown in Table 15 are based on several parameters with different metrics to validate that the BT P2P system is scalable. The results are also consistent with the arguments in section 8.3 and illustration of Figure 23 illustrating that average latency increases when there are more users searching for the file. Here the following parameters and metrics were used to simulate the P2P environment:

- network size: 1,000
- peerset size: 900
- maximum growth: 100
- file size: 100 MB
- link speed: 200 kb/s
- duplicate request: 1
- fragment size: 32 KB, 64 KB and 128 KB
- seeder percentage: 1
- peer percentage: 90, 95 and 99

Table 15: Simulation results of 90, 95 and 99% peers, 200 kb/s links, 100 MB file

Link Capacity: 200 kb/s	% Downloading	% Downloading	% Downloading
Piece Size	99%	95%	90%
32 KB	14190000	7240000	10920000
64 KB	5650000	5750000	5330000
128 KB	3160000	2910000	2750000

Seeder: 1%

The values of Table 15 are plotted in a graph and illustrated in Figure 25.

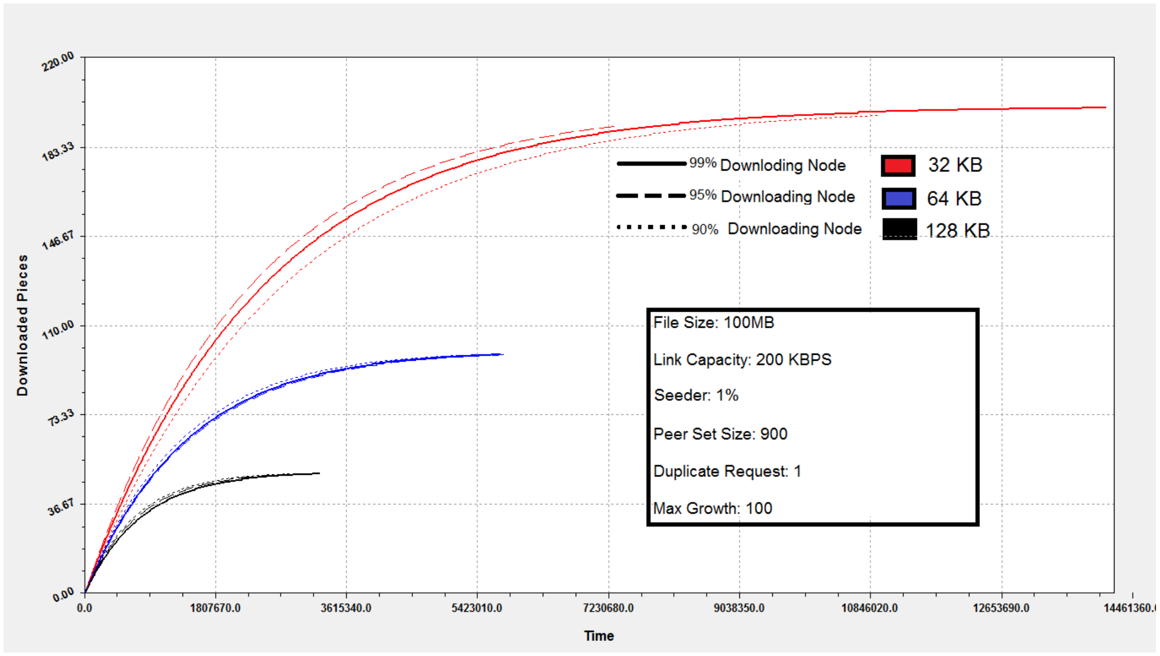

Figure 25: Comparison with different fragment sizes and network link capacity

It is clear from the results of Table 15 and Figure 25 that total time to download the 100 MB file increases when the percentage of peers of the network grow from 90 to 99.

8.6 File and Fragment Size Evaluation

The parameters of section 8.5 were fine-tuned and simulated with different metrics to determine combination of what values enable optimized performance. Here are the values:

- network size: 100
- peerset size: 50
- maximum growth: 20
- swarm size: 80
- file size: 50 MB
- link speed: 50 kb/s

- duplicate request: 1
- fragment size: 32 KB, 64 KB and 128 KB
- seeder percentage: 1
- peer percentage: 1, 5 and 10

Table 16 has the values for sharing a 50 MB file in 32 KB, 64 KB and 128 KB fragments by the participating nodes using 50 kb/s data rate links.

Table 16: Simulation results of 1, 5 and 10% peers, 50 kb/s links, 50 MB file

Link Capacity: 50 kb/s	% Downloading	% Downloading	% Downloading
Piece Size	1%	5%	10%
32 KB	200000	210000	240000
64 KB	220000	160000	200000
128 KB	130000	160000	170000

Seeder: 1%

The values of Table 16 are plotted in a graph and illustrated in Figure 26.

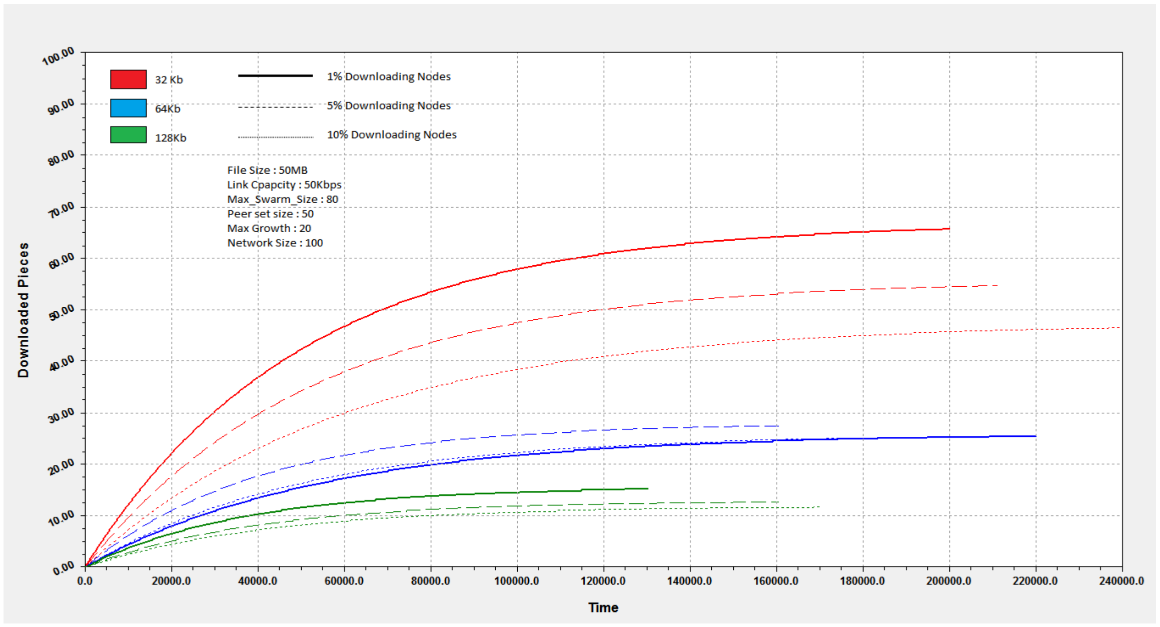

Figure 26: Time to share 50 MB file using 50 kb/s links in 100 node network size comparing 32, 64 and 128 KB fragments

Based on the results of Table 16 and Figure 26, it is evident that 128 KB fragment size fares significantly better than 32 KB and 64 KB fragments. This is because it generates less control traffic since there are fewer blocks and fragments of the large file.

8.7 Physical Link Data Rate Evaluation

The parameters and metrics of section 8.6 were used to simulate the P2P environment, but with 100 kb/s physical link data rate.

Table 17 has the values for sharing a 50 MB file by the participating nodes in 32 KB, 64 KB and 128 KB fragments using 100 kb/s data rate links.

Table 17: Simulation results of 1, 5 and 10% peers, 100 kb/s links, 50 MB file

Link Capacity: 100 kb/s	% Downloading	% Downloading	% Downloading
Piece Size	1%	5%	10%
32 KB	130000	490000	160000
64 KB	340000	210000	210000
128 KB	860000	250000	110000

Seeder: 1%

The values of Table 17 are plotted in a graph and illustrated in Figure 27.

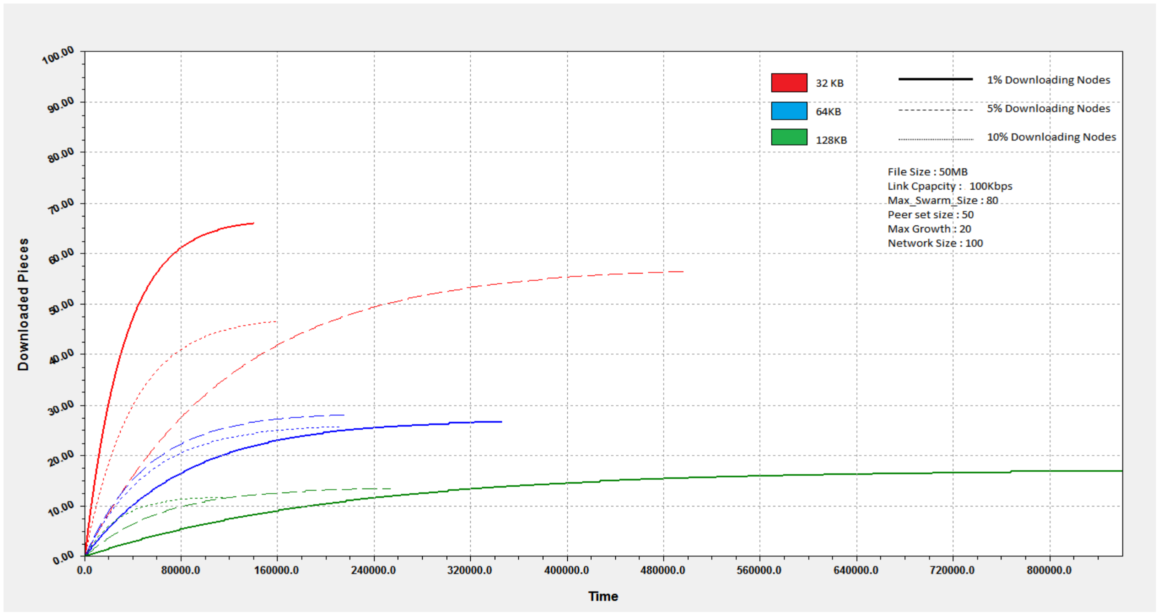

Figure 27: Time to share 50 MB file using 100 kb/s links in 100 node network size comparing 32, 64 and 128 KB fragments

Here we see that 128 KB fragments take less time to download the 50 MB file when 10% of the peers in the network are trying for the file. The assumption here is when 10 peers (10% of 100 nodes) try for the file, the possibility of one of the 10 peers (leechers) becoming a seed is greater than when 1% or 5% try for the file. When a leecher becomes a seed, it only uploads as opposed to both download and upload. This increases link utilization of the peer and enable other leechers to complete the file download faster to become a seed, and further assist other remaining leechers to complete their downloads quickly and become seeds. For 1% peers downloading, 32 KB and 64 KB does better because only 1 peer is interested in the file. This does not generate as many control traffic as it does when more peers are sharing the file content.

Figure 28 shows the number of pieces Node 1 downloads and uploads during the simulation. This illustrates the system achieves fairness by having nodes in the network upload the pieces they download to other peers that need them. After downloading all the pieces, the node becomes a seed and only uploads pieces to other peers.

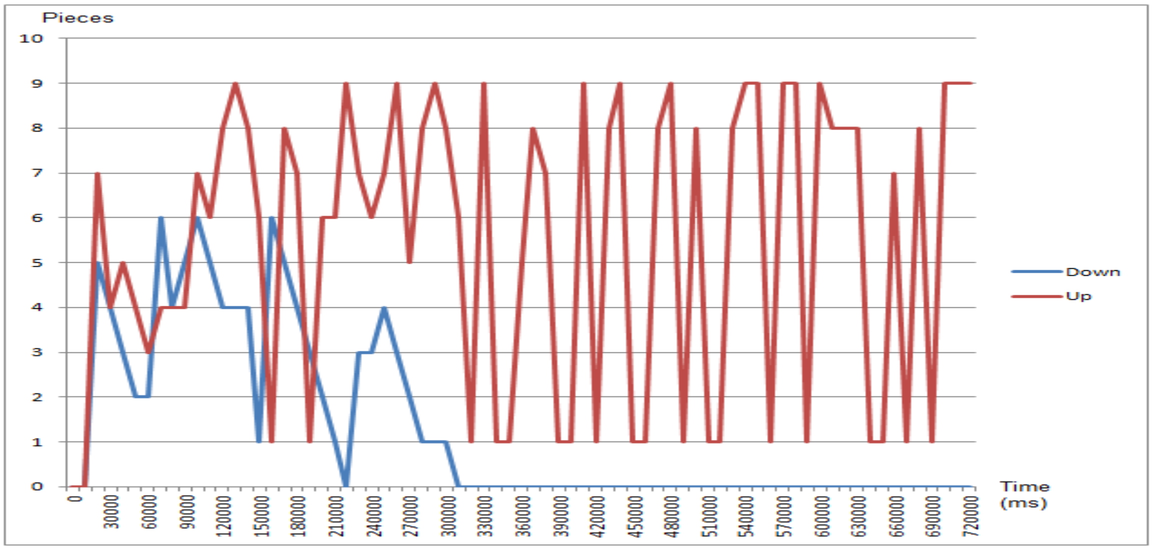

Figure 28: Download and upload values of Node 1

On a side note, nodes with higher link capacities have better downloading rates, and are unchoked more often than the nodes with slower links. This can create an unfair scenario in a heterogeneous environment where there are large disparities in the link capacity of nodes. Prior work show nodes with better downloading rates tend to complete downloading of the entire file faster and leave the network [122]. The selected architecture will be deployed in developing countries where there is little disparity in link capacities because the majority of nodes has limited capacity network links, and so would not encounter unfair file sharing conditions as would occur in developed countries.

8.8 Performance Optimization by Peer Size Percentage

To further optimize the system, simulated similar environment with different network, peerset, and percentage of peer (leecher) sizes. It was determined via the simulations of sections 8.6 and 8.7, 10% leechers perform well when they use 128 KB fragments to share a 50 MB file with 50 kb/s and 100 kb/s physical data rate links. Since 10% leechers of a network size of 100 is quite a small number (10), it was important to determine the results when there are more leechers. That is, 80% or 90% of leechers in a network of 100 peers.

This was made possible by simulating the same environment as section 8.6 with 50 kb/s link capacities. 100 MB files and 100 kb/s links were also simulated, but rejected from this analysis because its performance was not practical.

Table 18 has the values for sharing a 50 MB file by the participating nodes in 32 KB, 64 KB and 128 KB fragments using 50 kb/s data rate links.

Table 18: Simulation results of 80 and 90% peers, 50 kb/s links, 50 MB file

Link Capacity: 50 kb/s	% Downloading	% Downloading
Piece Size	80%	90%
32 KB	810000	1410000
64 KB	440000	1540000
128 KB	360000	760000

Seeder: 1%

The values of Table 18 are plotted in a graph and illustrated in Figure 29.

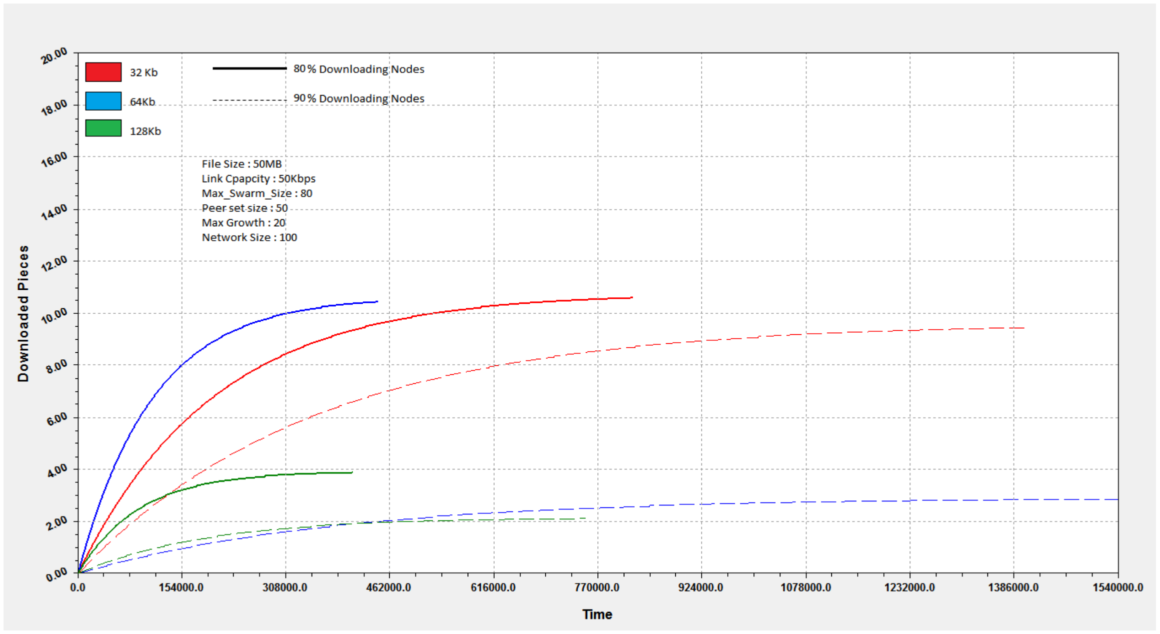

Figure 29: Time to share 50 MB file using 50 kb/s links in 100 node network size comparing 32, 64 and 128 KB fragments

Here, we see for both 80% and 90% leechers in a network of 100 nodes, 128 KB fragments perform better than 32 KB and 64 KB fragments. It takes 760 seconds to download the 50 MB file by 90 leechers and 360 seconds by 80 leechers with 50 kb/s data rate links.

Best-of-the-best results of 1, 10 and 80 percent peers downloading in a network size of 100 nodes is presented in Table 19 and plotted in Figure 30.

Table 19: Optimum performance with 100 nodes

% Downloading	File (MB)	Link (kb/s)	Fragment (KB)	Time (ms)
1	50	50	128	130000
10	50	100	32	160000
10	50	100	128	110000
80	50	50	128	360000

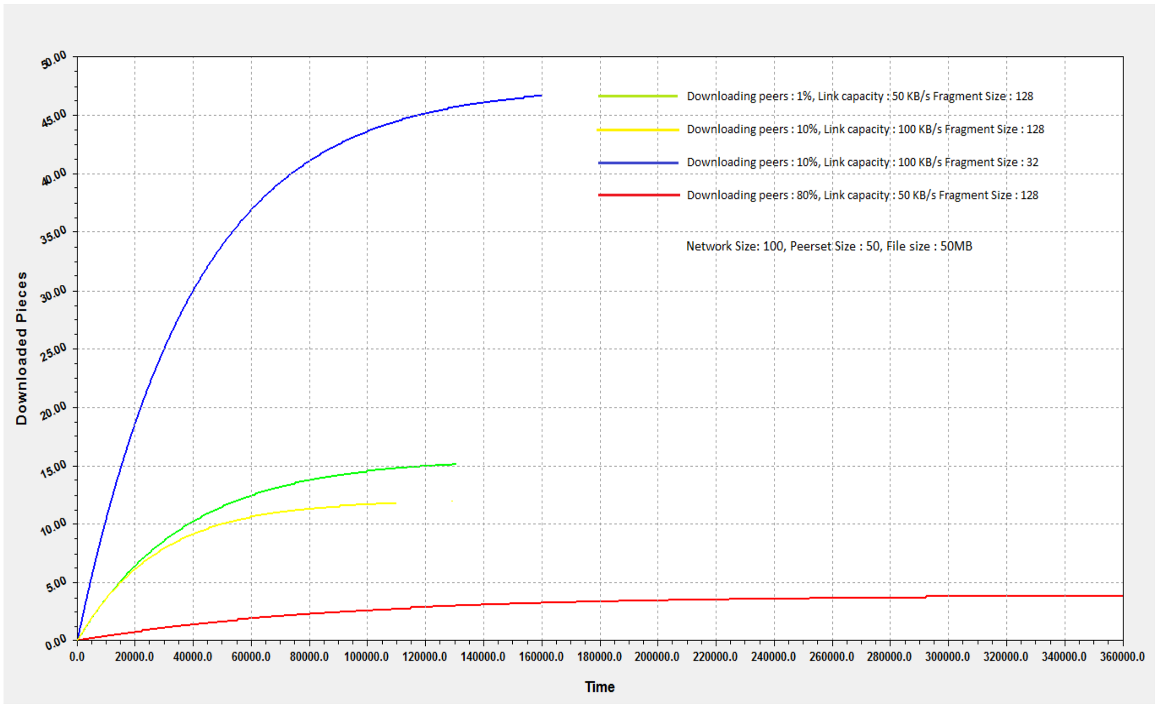

Figure 30: Best-of-the-best results of 100 nodes

Is it possible to improve this any further? Different combinations of parameters were next simulated using only 100 kb/s data rate links.

8.9 Evaluation of Larger Network, Peerset and File Sizes with Small Percentage of Peers

The next approach was to increase the size of the network so percentage of peers could be reduced, yet maintaining close to similar number of leechers.

The following parameters were fine-tuned and simulated with different metrics of file and fragment sizes, and peer percentages to determine the combination of values that improved performance further. The parameters and metric values are:

- network size: 500
- peerset size: 100
- maximum growth: 20
- swarm size: 80
- file size: 100 MB
- link speed: 100 kb/s
- duplicate request: 1
- fragment size: 32 KB, 64 KB and 128 KB
- seeder percentage: 1
- peer percentage: 15, and 20

With 15% peers in a network of 500 peers, we have 75 leechers, and with 20% peers we have 100 leechers, which is close to the 80 and 90 leechers of the experiments done in section 8.8.

Table 20 has the values for sharing a 100 MB file in 32 KB, 64 KB and 128 KB fragments by the participating nodes using 100 kb/s data rate links.

Table 20: Simulation results of 15 and 20% peers, 100 kb/s links, 100 MB file

Link Capacity: 100 kb/s	% Downloading	% Downloading
Piece Size	15%	20%
32 KB	170000	100000
64 KB	240000	170000
128 KB	220000	120000

Seeder: 1%

The values of Table 20 are plotted in a graph and illustrated in Figure 31.

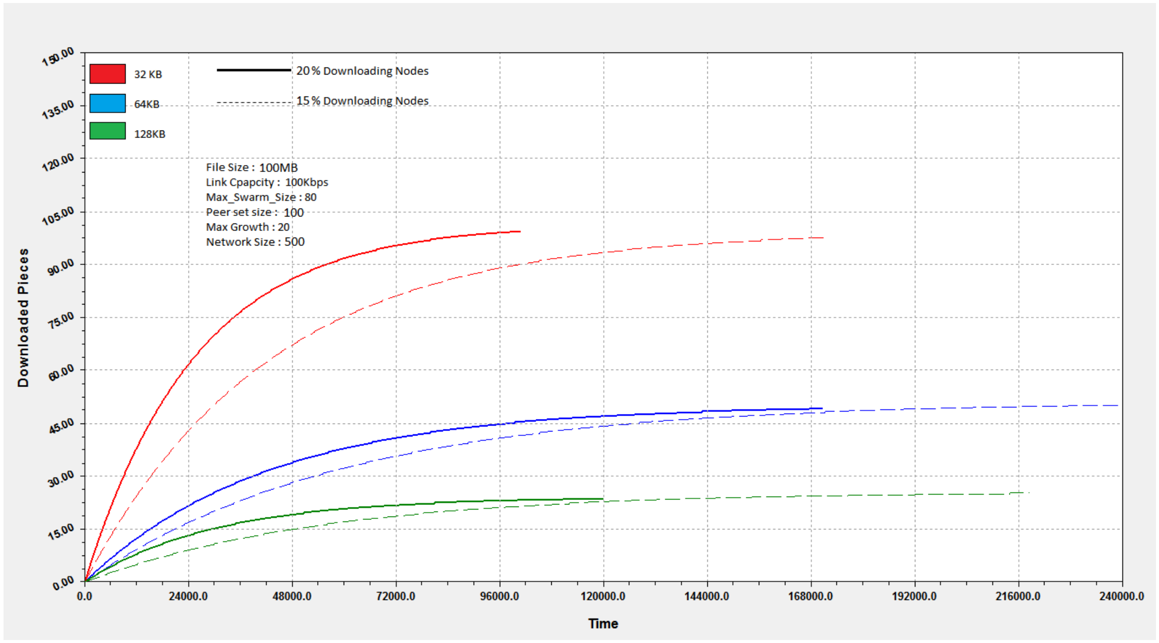

Figure 31: Time to share 100 MB file using 100 kb/s links in 500 node network size comparing 32, 64 and 128 KB fragments

From the results of this section, we see 32 KB fragments perform slightly better than 128 KB fragments. These results helped determine both 32 KB and 128 KB fragments used to download and share a 100 MB file using 100 kb/s physical data rate links in a network of 500 peers by 20% leechers achieve best performance.

Table 21 has the optimum performance attained via combination of different parameters in a network size of 500 nodes, and summary graph of the best-of-the-best results is presented in Figure 32.

Table 21: Optimum performance with 500 nodes

% Downloading	File (MB)	Link (kb/s)	Fragment (KB)	Time (ms)
15	50	100	128	110000
20	100	100	32	100000
20	100	100	128	120000

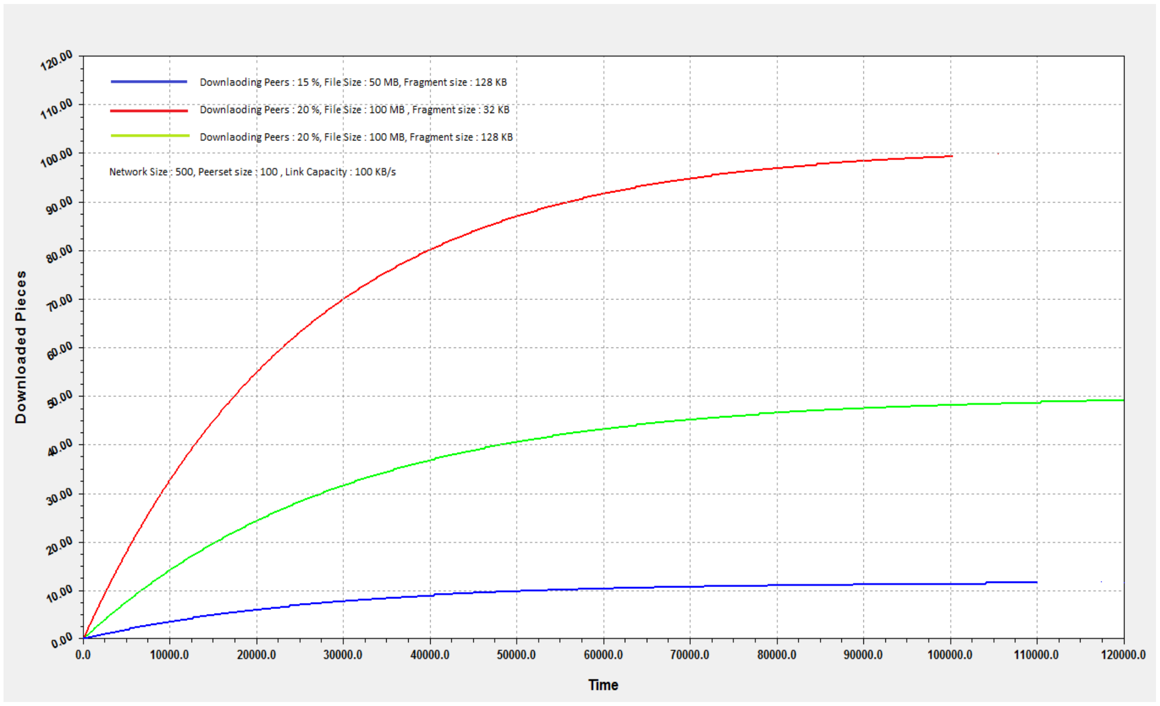

Figure 32: Best-of-the-best results of 500 nodes

8.10 Impact of Seeders on Performance

To verify the impact seeders have on the system, simulations were run with the following parameters using different percentage of peers and seeders:

- network size: 1,000
- peerset size: 900
- maximum growth: 100
- file size: 50 MB
- link speed: 200 kb/s
- duplicate request: 1
- fragment size: 32 KB, 64 KB and 128 KB
- seeder percentage: 1, 2 and 5
- peer percentage: 70, 80, 90, 95 and 99

Table 22: Simulation results of 99, 95, 90, 80 and 70% peers and 1% seeders, 200 kb/s links, 100 MB file

Link Capacity: 200 kb/s	% Peers	% Peers	% Peers	% Peers	% Peers
Piece Size	99%	95%	90%	80%	70%
32 KB	5810000	5390000	5150000	4845018	4760000
64 KB	3166018	2830000	2590000	2520000	2390000
128 KB	1760000	1522016	1510000	1285016	1310000

Seeder: 1%

Table 23: Simulation results of 99, 95, 90, 80 and 70% peers and 2% seeders, 200 kb/s links, 100 MB file

Link Capacity: 200 kb/s	% Peers	% Peers	% Peers	% Peers	% Peers
Piece Size	99%	95%	90%	80%	70%
32 KB	5554016	5310000	4910000	4710000	4770000
64 KB	2885016	2800000	2440000	2487016	2360000
128 KB	1680000	1460000	1470000	1300000	1190000

Seeder: 2%

Table 24: Simulation results of 99, 95, 90, 80 and 70% peers and 5% seeders, 200 kb/s links, 100 MB file

Link Capacity: 200 kb/s	% Peers	% Peers	% Peers	% Peers	% Peers
Piece Size	99%	95%	90%	80%	70%
32 KB	5390000	4963004	4900000	4490000	4610000
64 KB	2551016	2690000	2580000	2270000	2200000
128 KB	1480000	1330000	1320000	1230000	1240000

Seeder: 5%

Based on the results of Tables 22, 23 and 24, it is evident that as percentage of seeders goes up at the start of the simulation, total time to download the entire file by the peers is reduced. It is interesting to note from these tests that there is not a significant difference in performance. This is because once leechers download the entire file, they also become seeders, so a larger population of seeders only help peers with faster download during the early phase of the simulation. As the network becomes more distributed with converted seeders, there is less dependence on the initial seeders. This interesting observation from the simulation results has helped determine there is little impact on performance optimization by the initial number of seeders.

8.11 Peer Dynamics

As demonstrated in section 8.9, a network of 500 peers achieves best performance for this work, but it is critical to determine whether the system is resilient during churn. Several churn rates using the parameters and metric values of section 8.9 with 100 MB file, 128 KB fragment and 20% peers trying for the file, were simulated to conclude the system scales well when there is a flash crowd limit set to 10% of total number of peers in the network. That is, the system is able to sustain network instability with peers joining and leaving the network at a time in groups of 5 to 50 peers every 10 seconds because the total download time increases from 150 seconds with churn rate of 5 to 400 seconds with churn rate of 50 peers.

Table 25 has the results of the simulations.

Table 25: Simulated results of peer dynamics

Peer Dynamics	Dynamics Freq	Time
5:5	10000	150000
50:50	10000	400000

8.12 Impact of Duplicate Requests

It is possible to control the number of duplicate *request* messages that are sent to different peers by a peer after an *unchoke* for the same block is received. For example, if the value of duplicate requests is 2, a node may request every block to 2 different neighbors. The default value used so far was 1.

Duplicate request counts of 1, 2, 4 and 8 are simulated keeping the same parameters of section 8.9 with 100 MB file, 128 KB fragment and 20% peers trying for the file, to evaluate the impact it has on performance.

Table 26: Simulation results of duplicate request packets

Number of Duplicate Requests	Time
1	12000
2	13000
4	16000
8	18000

From the results of Table 26, it was determined duplicate packet requests have a negative impact on performance. This is why the duplicate request message will be set to one (1) and not zero (0), because when it is set to zero, the peers are not able to complete a download of the entire file since they are unable to request for the same message from another node.

8.13 Swarm Size Evaluation

Here different swarm sizes are simulated keeping the other parameters of section 8.9 with 100 MB file, 128 KB fragment and 20% peers trying for the file. Here are the parameters and metric values that were used:

- network size: 500

- peerset size: 100
- maximum growth: 20
- swarm size: 110, 120, 150 and 200
- file size: 100 MB
- link speed: 100 kb/s
- duplicate request: 1
- fragment size: 128 KB
- seeder percentage: 1
- peer percentage: 20

Table 27: Simulation results of swarm sizes

Swarm Size	Time
110	100000
120	100000
150	90000
200	50000

From the simulation results of Table 27, it is evident that the network can be optimized if swarm size of 200 is used. Beyond 200, there is no performance gain, and it would lead to unnecessary wastage of peer resources because each peer would need to keep *n* (200+) number of peers' information in its cache. It is determined from this experiment that swarm size of 200 achieves the best performance and will be used in the selected architecture along with the parameters and metrics itemized in this section above.

8.14 *n*-Tier Hierarchical Training Model

The *n*-tier hierarchical model described here may be applied in any developing nation, but the example presented in this section focuses on Bangladesh, a developing country that does not have the physical infrastructure to offer high speed wireless Internet service. ISPs offer "broadband"

Internet service with data rates up to 500 kb/s using physical cable infrastructures, but this service is available only to households in large metropolitan areas. There are only six major cities in six Divisions or Districts in the country. There are 481 Upazilas, of which 121 are in Dhaka Division, 99 in Chittagong Division, 121 in Rajshahi Division, 59 in Khulna Division, 38 in Sylhet Division, and 40 in Barisal Division. On average, population of each Upazila is approximately 300,000 [128]. Upazilas are small townships located in Districts or Divisions. Districts are larger areas similar to the concept of a state in the United States. Upazilas can be viewed as a county within a state. There are hundreds of small villages in each Upazila.

There has been major advancement with wireless communication in developing countries because they were able to leapfrog the developed countries leveraging wireless technology, and significantly improve their data communications physical infrastructure to meet the growing needs of their vast populations. There are about 90 million users of cellular phones [21] and 617,300 Internet users [5] in Bangladesh. Unlike developed nations, wireless Internet service is provided via EDGE cellular networks that are able to offer up to 100 kb/s data bit rate. There has been tremendous growth of wireless networks, and illustrated in Figure 33. The data rate of EDGE is not adequate enough to support media-rich content like video and annotated power point slides, but is fine for voice.

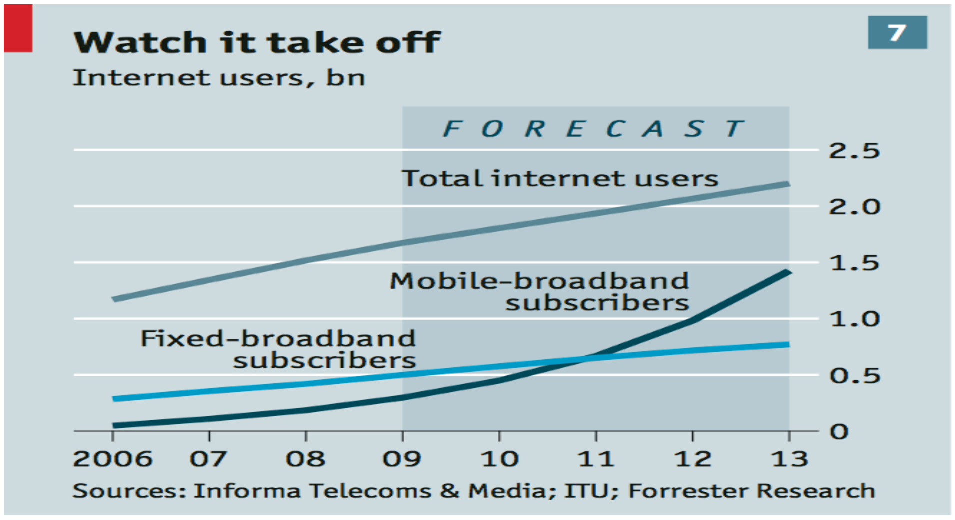

Figure 33: Growth of Internet use: wired vs. wireless worldwide[14], [129]

IT-tool developers in the developed countries are busy enhancing features of the LMSs and SDETs to make them more user-friendly. They are not keen on releasing a version of the tool that would operate in low data rate networks of the developing world. This is because their primary audience is in the developed countries where low speed networks are not an issue. Going forward, we need to teach with what we have in hand at present, because time is of the essence. To do so, a layered hierarchy described below and illustrated in Figure 34, is proposed to facilitate online instruction environments with low network capacity service providers.

Internet-capable mobile phones and smartphones are clearly the instruction delivery platform of the future [14]. Tablet computers have been proliferating in developed countries and due to their cost, it is yet to be seen whether these devices will be used as the instruction platform. It is quite unlikely these devices will be used in the developing world for online instructions. Smartphones are cost-prohibitive for the poor and majority of students lacking basic education cannot afford to own these smart devices. It would be difficult to reach these students from distant locations via the Internet.

The other option is to have them trained in classrooms at their location by local instructors.

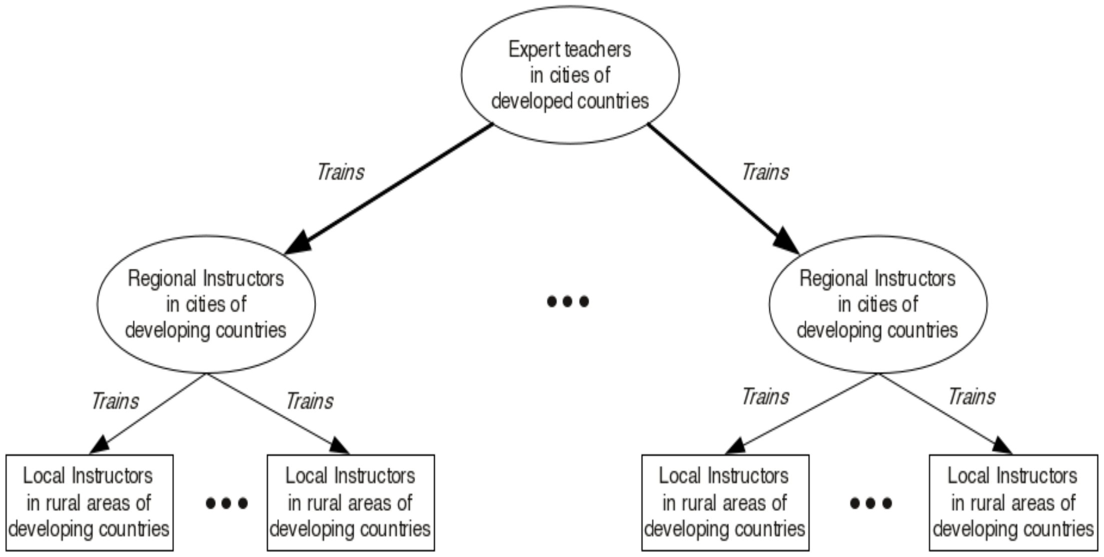
Expert teachers in cities of developed countries
Trains
Trains
Regional Instructors in cities of developing countries
Regional Instructors in cities of developing countries
Trains
Trains
Trains
Trains
Local Instructors in rural areas of developing countries
Local Instructors in rural areas of developing countries
Local Instructors in rural areas of developing countries
Local Instructors in rural areas of developing countries

Figure 34: *n*-tier hierarchical training model

The problem is that there is a huge demand for quality teachers in developing countries because good teachers are hard to find there. On a positive note, quality teachers can be teaching from any part of the world. There may be a good supply of quality teachers in an urban area, but only a few in remote rural areas [31]. Using synchronous and asynchronous distance education delivery methods, in addition to teaching students, experienced teachers can also train the trainers, where quality teachers would train inexperienced teachers in remote areas to better prepare them to teach local students in classrooms. This *n*-tier hierarchical model would enable students with no access to smartphones, computers and/or Internet service to learn from local instructors in a classroom setting. Synchronous DE delivery method may also be leveraged to offer tutorials to remote teachers on using distance education tools, so they are able to use these tools that would be needed to train local instructors, and teach students in remote areas via DE delivery method. It is important to train local instructors in using the technology because researchers argue it is critical to train the instructors to use computers, emerging networking technologies and distance education tools for DE [17].

With the hybrid learning model, web-based DE is blended with face-to-face teaching sessions so that online education is at par with the quality of campus programs. This is because research studies show classroom student performance is better with home assignments and projects, whereas web-based DE students perform better with exams [17]. Hybrid DE is also more effective because there is periodic face-to-face interaction between the instructor and students. Quality instructors can train remote local instructors via web-based DE delivery methods to enable local instructors to teach students in live classrooms at the remote location.

As established in Chapter 1, the goal of the selected architecture is to train the *regional* instructors who are located at large metropolitan cities of the six Divisions or Districts by the bright, experienced and motivated *expert* instructors located at cities in developed nations. Once they are trained, the regional instructors would then train the *local* instructors residing in Upazilas. Our target audience is about 500 local instructors: one per Upazila who will be trained by the regional instructors via synchronous and asynchronous DE using limited capacity networks. My design chose 500 as a practical limit but in reality we will work with no more than 100 local instructors. Figure 34 illustrates the training process via the *n*-tier hierarchical training model. This model is used to train instructors at remote locations and is yet another secondary contribution of this work.

The *n*-tier hierarchical training model is also applicable for teaching students by instructors at tier-1 and tier-2 via both the synchronous and asynchronous DE delivery methods. Again, we chose to limit the target student audience to the practical number of 500. For example, the expert and regional instructors of tier-1 and tier-2, respectively, can engage in running synchronous and asynchronous DE classes to teach students located in remote Districts and/or Upazilas. The student population size has been limited to 500 because it would be very difficult for the instructor to communicate with more than 500 students during the same synchronous session. 500 students is itself a large number, but being ambitious, the hope is not more than ten students will ask questions at the same time.

It is important to note that network capacity not be as limited in the upper tiers of this model as they are in lower tiers, because the lower tier nodes are originating from either developed countries or metropolitan cities of the developing countries. Network capacity is not as abundant as one traverse down the tiers to the leaf nodes, because those nodes are at the remote locations that are dependent on cellular Internet service, primarily EDGE. Here is where the selected architecture becomes very useful and at the same time effective, because it would enable content delivery and exchange via limited capacity networks.

Figure 35 is notional showing the concept of data bit rate in developing countries as one moves farther away from the major cities to remote locations. The circles are color coded to show the outermost circle is farthest from a large metropolitan city and has little or no Internet service. Moving inward, the next area has very slow data bit rate Internet service and the innermost area has broadband Internet service. The *n*-tier hierarchical model may be tied to the data bit rate diagram where the top tier users are located in the area with high speed Internet service. As one traverses down the hierarchical model to lower level tiers, he or she is moving farther away from the center of the circle having to work with slower data bit rate Internet connections using EDGE cellular service.

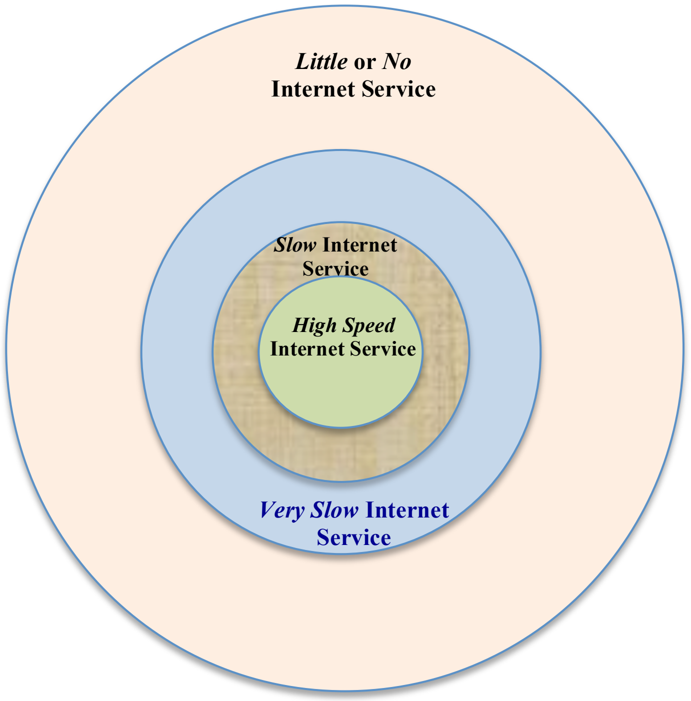
Little or *No* Internet Service
Slow Internet Service
High Speed Internet Service
Very Slow Internet Service

Figure 35: Notional data bit rate map

8.15 Conclusion

More than 200 combinations of parameters and metrics were simulated to determine the critical combinations and metric values for the selected architecture of this work. The following parameters and metrics with BT protocol have been determined to optimize the performance of the system, and selected for this architecture:

- network size: 500
- peerset size: 100
- maximum growth: 20
- swarm size: 200
- file size: 100 MB
- link speed: 100 kb/s
- duplicate request: 1
- fragment size: 32 and 128 KB
- seeder percentage: 1
- peer percentage: 20

CHAPTER 9: CONCLUSIONS, CONTRIBUTIONS AND FUTURE DIRECTIONS

9.1 Conclusions

This work focuses on DE and efficient content delivery and exchange via limited capacity networks using P2P systems and overlay networking. Chapter 1 introduces the concept of DE and presented facts on its effectiveness in comparison to live classroom instruction settings. Characteristics of asynchronous and synchronous DE are described and their advantages and disadvantages compared to validate a blended approach results in the most effective learning environment. The importance of education in developing countries is discussed and facts presented to show how DE can play a major role in raising the literacy rate there. There is a need for a synchronous distance learning tool that operates over low capacity data bit rate of 56 kb/s because in developing countries Internet service is poor due to limited network capacity. Since the Internet is the vehicle to deliver content to students, and limited capacity EDGE cellular service is the common infrastructure to connect to the Internet, there is a need for a network architecture to facilitate content delivery and exchange via low capacity cellular networks. As described in Chapter 5, the design of this architecture cannot use traditional pure client-server model or IP multicasting. The option is to use P2P distributed nodes in an overlay network that store content fragments and process search queries to enable DE content delivery and exchange via existing wireless infrastructure at least cost.

Chapter 2 presents the review of 18 structured and unstructured P2P systems and architectures, and chapter 3 discusses prior work of 12 emerging overlay technologies to determine BT as the suitable system to use as the platform for this architecture. Chapter 4 presents comparisons of six LMSs and five SDETs to conclude MIST/C as the most cost-effective, easy-to-use and simple distance education tool available in the market today because of its rich features and functionalities, it is open source and is able to operate over limited capacity networks.

Based on the requirements laid out in chapter 6, a framework of the architecture with examples and illustrations is presented in chapter 7 and serves as the basis for chapter 8, which presents a top-down ordered progression structure of simulated experiments, describing each test methodology. The simulated results of chapter 8 support comparison of many parameters and metrics to determine the best performance of the architecture. An *n*-tier hierarchical training model is developed and also presented in section 8.14 to show how it can be used with this architecture to train instructors and students using leaf nodes of the lower tiers at the remote locations that are dependent on low capacity data bit rate cellular Internet service.

The selected architecture may be used for any type of content delivery and exchange, but the focus has been on DE content because the goal of this work is to implement this solution and enable students living in rural areas of both developed and developing countries to get the proper education they need. Since quality teachers are scarce in those areas, DE content delivery and exchange can work as an alternative for those students to learn from quality teachers residing in cities far away. But as described in section 8.14, under-served students in the rural areas cannot afford to own a computer and have Internet service. The *n*-tier hierarchical training model can be used to train the junior local instructors of rural areas so they can gradually improve their teaching skills while continuing to teach the local students in live classrooms, and finally become quality teachers.

9.2 Contributions

The most significant contribution of this work was the discovery and integration of an effective architecture based on extensive research of emerging DE and Overlay networking technologies, and P2P systems. Three primary contributions and seven sub-contributions of this work are now discussed.

The first primary contribution was the review of eighteen P2P unstructured and structured, overlay multicast and DE architectures that are presented in chapter 2. This review concluded that unstructured systems will not meet the challenge laid out in Chapter 1 because they are not scalable due to their blind search mechanism, low fault tolerance and not having files available at all times. Structured systems use identifiers for exact match in their search process to be scalable but this is not a solution in transient node population. The overlay multicast architectures surveyed were not suitable candidates for the selected architecture because they do not scale well in limited capacity networks and churn.

The second primary contribution was the survey of twelve protocols, routing algorithms, overlay tree structures and replication strategies presented in chapter 3 to determine their capabilities in addressing problems surrounding network congestion, single point of failure and transient node population. The reviews of chapters 2 and 3 led to the selection of BT as the platform for the target architecture because it is effective with bulk file transfer using its swarming feature to distribute data to achieve better performance and resource utilization.

The third primary contribution was the determination of the critical combination of parameters and values to use collectively with BT for designing a solution that meets the minimum requirements of the target architecture. This was achieved by simulating experiments with different parameters and values, for analysis to conclude on design of the selected architecture.

There also are seven sub-contributions of this work. First, comparison of six LMSs: two developed by education institutions, and four by commercial entities; and five SDETs: two commercial products and three open source were done to select the most suitable DE application tool to use with the target architecture. MIST/C was selected for this purpose. This comparison and justification for selecting MIST/C is presented in chapter 4.

Second, a thorough analysis of the client-server architecture was done to show its drawbacks supplemented by simulated tests and draw to the conclusion it is not a suitable candidate for the target architecture. Next, problems with the traditional data communications model were presented to justify it is not suitable for this work also.

Sub-contributions four and five were based on the requirements of the architecture outlined in chapter 6. These were the design of a framework for the target architecture and the design of a four layer architecture.

The sixth and seventh sub-contributions stemmed from the third primary contribution to determine the effective system performance for the architecture. They were the design of a top-down ordered structure of simulation experiments to describe test methodologies and simulated set of experiments using different parameters and values for analysis.

9.3 Future Directions

The survey of chapter 4 helped determine MIST/C as the SDET to use with the selected architecture for DE content delivery and exchange. At present, MIST/C does not support smart device OS platforms. Section 8.14 validates there is huge growth of wireless mobile users worldwide. The main focus of LMS and SDET tool vendors has been on developed countries to enhance features and usability instead of looking into ways to enable the tools to operate in limited capacity networks because data bit rate is not an issue there as it is in developing countries. Since MIST/C is able to operate over 56 kb/s networks, it will be necessary to develop a version that can run on mobile devices because DE is rapidly expanding in the developing world where the primary mode of connection to the Internet is the cellular network. Internet-capable mobile phones and smartphones are clearly the instruction delivery platform of the future [14]. Considerable code updates of MIST/C will need to be made to support the Android OS platform.

MIST/C has the feature to record lectures but does not offer an easy control interface to edit the audiographics recording. This turns out to be a problem for the instructor because it becomes necessary to play through the entire lecture if a minor change is made during the following semester. This is another area of MIST/C where work can be done to enable audiographics editing to the extent one can edit in MIST/C, and also can edit the audiographics video.

This work has focused on developing countries, particularly Bangladesh due to my familiarity with it. The language in Bangladesh is Bengali and little of the population is fluent in English. We have been exploring avenues with foundations and the Government of Bangladesh to integrate MIST/C using the DE model with their education drive to help educate primary and secondary level students in remote areas. We demonstrated MIST/C to them and they are interested to initiate a pilot project, when support becomes available, to broaden their reach to the students where Internet service is available via slow data bit rate cellular service. To enable this, it will be necessary to have a Bengali language interface version of MIST/C.

In order to have a custom P2P network, it is necessary to write a software application that would establish access control and provide the features of existing P2P systems to the users and the whole system. This will be possible with torrents, tracker and web servers, and resources set up for the specific client application based on the BT protocol. This system will have its own format of torrent files and will only work with the custom software application. Only the authorized users of this network may participate in DE content delivery and exchange having the custom client application. The environment will have customized torrent files, and a TS to send data packets only to the users of this software. As with other TSs, this TS would only keep track of peers that are using the custom application. It will be necessary to have reliable connectivity to the TS as it would be the logical backbone of the custom software based authenticated network. The other important factor is the availability of the torrent file. There has to be a mechanism to deliver the torrent files to the client. For this, a simple WS would need to be provisioned from which users would receive the respective torrent file. Once the torrent is downloaded from the WS, it would launch via the custom POCD client application and redirect the user to the TS, which would provide details on the peers who are also sharing the same file. Live streaming with minor changes to BT [87] need to be added and tested with the custom client application during implementation. Similar custom software applications may be developed for purposes other than DE.

To address single point of failure vulnerability of the TS, another recommendation is to replace the TSs with SNs in the system. Since SNs are not dedicated machines and can leave the network at any time, backup and replication mechanism need to be in place to enhance the availability of torrents.

As described in chapter 1, most of the underserved children living in remote areas lack basic education, which is key for economic growth, better health and quality of life, productivity, higher well-being of women and improved government [11], [12]. Due to lack of competent teaching staff, there is a huge student to teacher ratio that slows enrollment of out of school children. DE can be used to train instructors in remote locations by experienced teachers in cities, and fill the gap of teacher shortages. These teachers can then teach the children in local classrooms in villages. Competent teachers can also use DE to teach students of a classroom in villages remotely. Most of the developing nations lack broadband Internet connectivity and are dependent on limited capacity cellular service. The POCD architecture can be used with MIST/C to enable digital content delivery and exchange via limited capacity networks, and meet the important need of imparting primary, secondary and IT education to the underserved.

REFERENCES

[1] S. Smith, “Examining the impact of synchronous video on distance education delivery and outcomes,” Rensselaer Polytechnic Institute, 2007.

[2] T. Russell, *The no significant difference phenomenon: a comparative research annotated bibliography on technology for DE*. North Carolina State University, 2001.

[3] “World Internet users and population stats,” *Internet World Stats*. [Online]. Available: http://www.internetworldstats.com/stats.htm. [Accessed: 01-Oct-2012].

[4] “Internet users and population stats for North America,” *Internet World Stats*. [Online]. Available: http://www.internetworldstats.com/stats14.htm#north. [Accessed: 12-Oct-2012].

[5] “Asia Internet usage and population,” *Internet World Stats*. [Online]. Available: http://www.internetworldstats.com/stats3.htm#asia. [Accessed: 02-Oct-2012].

[6] I. E. Allen and J. Seaman, *Learning on demand: Online education in the United States, 2009*. Sloan Consortium Needham, MA, 2010.

[7] B. Parsad and L. Lewis, “DE at degree-granting postsecondary institutions: 2006-2007 (NCES 2009-044),” National Center for Education Statistics, Institute of Education Sciences, U.S. Department of Education, Washington, DC, Dec. 2008.

[8] M. Islam and A. Gronlund, “Digital Bangladesh- a change we can believe in?,” in *2nd International Conference on Electronic Government and the Information Systems Perspective*, Toulouse, France, 2011.

[9] *The millennium development goals report*. New York, United States: United Nations Department of Economic and Social Affairs, 2010.

[10] “Education,” *ONE*, 2012. [Online]. Available: http://www.one.org/c/us/issuebrief/93. [Accessed: 18-Dec-2012].

[11] K. R. Islam and C. Snow, “A cost-effective distributed architecture to enable distance education over emerging wireless technologies,” in *10th Annual Conference on Information Technology Education*, Fairfax, Virginia, 2009, pp. 182–188.

[12] K. R. Islam and C. M. Snow, “An architecture for delivery of distance education in developing countries,” in *12th Annual Conference on Information Technology Education*, West Point, New York, 2011, pp. 215–220.

[13] “Global targets, local ingenuity,” *The Economist*, 22-Sep-2010. [Online]. Available: http://www.economist.com/node/17090934. [Accessed: 05-Oct-2012].

[14] C. Snow and K. Islam, “MDG 2: Can IT save us?,” in *13th Annual Conference on Information Technology Education*, Calgary, Canada, 2012.

[15] K. E. Paprock, “The digital divide in developing countries: A case for distance education,” *Systemics, Cybernetics and Informatics*, vol. 4, no. 6, pp. 185–83, 2006.

[16] C. Snow, J. M. Pullen, and P. McAndrews, “Network EducationWare: an open-source web-based system for synchronous distance education,” *IEEE Transactions on Education*, vol. 48, no. 4, pp. 705–712, 2005.

[17] M. Hentea, M. J. Shea, and L. Pennington, “A perspective on fulfilling the expectations of distance education,” in *4th Conference on Information Technology Curriculum*, Lafayette, Indiana, 2003, pp. 160–167.

[18] J. M. Pullen, N. K. Clark, and P. M. McAndrews, “MIST/C: open source software for hybrid classroom and online teaching,” in *Technology for Education/758: Software Engineering and Applications*, Dallas, Texas, 2011.

[19] J. M. Pullen and N. K. Clark, “Moodle-integrated open source synchronous teaching,” in *16th Annual Joint Conference on Innovation and Technology in Computer Science Education*, Darmstadt, Germany, 2011, pp. 353–353.

[20] Y. Kim, T. Kelly, and S. Raja, “Building broadband: strategies and policies for the developing world,” The World Bank, Washington, DC, Jan. 2010.

[21] “Bangladesh’s cell phone subscribers cross 70 million in January,” *Bangladesh Business News*, Dhaka, Bangladesh, 23-Feb-2012.

[22] A. Chen and D. Chung, “Bangladesh b’cast & cable TV services severely disrupted by interference,” *The Wall Street Journal*, 14-Jul-2012.

[23] K. German, “Sprint unveils first 4G phone,” *CNET*, 23-May-2010. [Online]. Available: http://reviews.cnet.com/8301-12261_7-20000998-10356022.html. [Accessed: 14-Jul-2012].

[24] S. Segan, “WiMAX vs. LTE: Should You Switch?,” *PC Magazine*, 16-May-2012.

[25] “Inmarsat BGAN,” *L3 Communications*. [Online]. Available: http://www.globalcoms.com/products_satellite_bgan.asp. [Accessed: 14-Jul-2012].

[26] “Broadband Global Area Network,” *Wikipedia*. [Online]. Available: http://en.wikipedia.org/wiki/Broadband_Global_Area_Network. [Accessed: 14-Jul-2012].

[27] H. Singer, “Are wireless data prices in the United States too high?,” *Forbes*, 19-Oct-2012. [Online]. Available: http://www.forbes.com/sites/halsinger/2012/10/19/are-4g-lte-prices-in-the-united-states-too-high/. [Accessed: 20-Feb-2013].

[28] R. Llamas and W. Stofega, “Worldwide smartphone 2012-2016 forecast and analysis,” Mar-2012.

[29] R. Anderson, T. VanDeGrift, and F. Videon, “Videoconferencing and presentation supprt for synchronous distance learning,” in *33rd ASEE/IEEE Frontiers in Education Conference*, Westminster, Colorado, 2003.

[30] S. Hrastinski, “Asynchronous and synchronous distance learning,” *Educause Quarterly*, vol. 31, no. 4, pp. 51–55, 2008.

[31] W. Habib, “IT education in a mess,” *The Daily Star*, Dhaka, Bangladesh, 09-Apr-2011.

[32] A. Dymond, “Telecommunications challenges in developing countries,” The World Bank, Washington, DC, 27, 2004.

[33] J. Aker and I. Mbiti, “Mobile phones and economic development in Africa,” *Journal of Economic Perspectives*, vol. 24, no. 3, pp. 207–232, 2010.

[34] S. Ratnasamy, A. Ermolinsky, and S. Shenker, “Revisiting IP multicast,” in *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Pisa, Italy, 2006, pp. 15–26.

[35] K. Leibnitz, T. Hobfeld, N. Wakamiya, and M. Murata, “Peer-to-peer vs. client/server: Reliability and efficiency of a content distribution service,” in *20th International Teletraffic Conference on Managing Traffic Performance in Converged Networks*, Ottawa, Canada, 2007, pp. 1161–1172.

[36] A. Delis and N. Roussopoulos, “Performance and scalability of client-server database architectures,” in *18th International Conference on Very Large Data Bases*, Vancouver, Canada, 1992, pp. 610–610.

[37] D. Stutzbach and R. Rejaie, “Understanding churn in peer-to-peer networks,” in *6th Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Pisa, Italy, 2006, pp. 189–202.

[38] S. Androutsellis-Theotokis and D. Spinellis, “A survey of peer-to-peer content distribution technologies,” *ACM Computing Surveys (CSUR)*, vol. 36, no. 4, pp. 335–371, 2004.

[39] D. Schoder and K. Fischbach, “Peer-to-peer prospects,” *Communications of the ACM*, vol. 46, no. 2, pp. 27–29, 2003.

[40] M. Ripeanu, I. Foster, and A. Iamnitchi, “Mapping the gnutella network: properties of large-scale peer-to-peer systems and implications for system design,” *IEEE Internet Computing Journal (special issue on peer-to-peer networking)*, vol. 6(1), 2002.

[42] J. Liang, R. Kumar, and K. W. Ross, “Understanding kazaa,” Polytechnic University, New Yprk, 2004.

[43] M. Waldman, A. D. Rubin, and L. F. Cranor, “The architecture of robust publishing systems,” *ACM Transactions on Internet Technology (TOIT)*, vol. 1, no. 2, pp. 199–230, 2001.

[44] R. Dingledine, M. Freedman, and D. Molnar, “The free haven project: Distributed anonymous storage service,” in *Workshop on Design Issues in Anonymity and Unobservability*, 2001, pp. 67–95.

[45] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, “Chord: a scalable peer-to-peer lookup protocol for internet applications,” *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, pp. 17–32, 2003.

[46] B. Cohen, “Incentives build robustness in BitTorrent,” in *2nd International Workshop on Peer-to-Peer Systems*, Berkeley, California, 2003.

[47] W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmér, and T. Risch, "Edutella: a P2P networking infrastructure based on RDF," in *11th international conference on World Wide Web*, Honolulu, Hawaii, 2002, pp. 604–615.

[48] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making gnutella-like p2p systems scalable," in *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Karlsruhe, Germany, 2003, pp. 407–418.

[49] A. R. Bharambe, V. . Padmanabhan, and C. Herley, "Analyzing and improving a BitTorrent networks performance mechanisms," in *25th IEEE international conference on computer communications*, Barcelona, Spain, 2006.

[50] D. Qiu and R. Srikant, "Modeling and performance analysis of bittorrent-like peer-to-peer networks," in *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Portland, Oregon, 2004.

[51] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and replication in unstructured peer-to-peer networks," in *16th international conference on Supercomputing*, New York, United States, 2002, pp. 84–95.

[52] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, San Diego, California, 2001, pp. 161–172.

[53] P. Druschel and A. Rowstron, "PAST: A large-scale, persistent peer-to-peer storage utility," in *8th Workshop on Hot Topics in Operating Systems*, Oberbayern, Germany, 2001, pp. 75–80.

[54] B. Y. Zhao, J. Kubiatowicz, and A. D. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing," University of California, 2001.

[55] I. Clarke, O. Sandberg, B. Wiley, and T. Hong, "Freenet: A distributed anonymous information storage and retrieval system," in *Designing Privacy Enhancing Technologies*, Berkeley, California, 2001, vol. 9, pp. 46–66.

[56] I. Clarke, S. G. Miller, T. W. Hong, O. Sandberg, and B. Wiley, "Protecting free expression online with Freenet," *IEEE Internet Computing*, vol. 6, no. 1, pp. 40–49, 2002.

[57] S. Saroiu, K. Gummadi, and S. Gribble, “Exploring the design space of distributed and peer-to-peer systems: comparing the web, TRIAD and Chord/CFS,” in *1st International Workshop on Peer-to-Peer Systems (IPTPS) 2002*, Cambridge, MA, USA, 2002.

[58] C. G. Plaxton, R. Rajaraman, and A. W. Richa, “Accessing nearby copies of replicated objects in a distributed environment,” *Theory of Computing Systems*, vol. 32, no. 3, pp. 241–280, 1999.

[59] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, and others, “Oceanstore: an architecture for global-scale persistent storage,” *ACM Sigplan Notices*, vol. 35, no. 11, pp. 190–201, 2000.

[60] L. Lao, J. H. Cui, M. Gerla, and S. Chen, “A scalable overlay multicast architecture for large-scale applications,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 4, pp. 449–459, 2007.

[61] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, “Scalable application layer multicast,” in *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Pittsburgh, Pennsylvania, USA, 2002, vol. 32.

[62] Y. Chu, S. G. Rao, S. Seshan, and H. Zhang, “A case for end system multicast,” *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 8, pp. 1456–1471, 2002.

[63] G. Kwon and J. W. Byers, “ROMA: Reliable overlay multicast with loosely coupled TCP connections,” in *23rd Annual Joint Conference of the IEEE Computer and Communications Societies*, Hong Kong, 2004, vol. 1.

[64] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and others, “Overcast: reliable multicasting with on overlay network,” in *4th Conference on Symposium on Operating System Design & Implementation*, San Diego, California, 2000, vol. 4, pp. 14–14.

[65] H. W. Holbrook and D. R. Cheriton, “IP multicast channels: EXPRESS support for large-scale single-source applications,” *ACM SIGCOMM Computer Communication Review*, vol. 29, pp. 65–78, 1999.

[66] M. F. Kaashoek, R. van Renesse, H. Van Staveren, and A. S. Tanenbaum, “FLIP: an Internetwork Protocol for Supporting Distributed Systems,” *ACM SIGOPS Operating Systems Review*, vol. 26, no. 2, p. 29, 1992.

[67] G. Fox and S. Pallickara, “JMS compliance in the Narada event brokering system,” in *International Conference on Internet Computing*, Las Vegas, Nevada, 2002, pp. 391–402.

[68] S. Pallickara and G. Fox, “NaradaBrokering: a distributed middleware framework and architecture for enabling durable peer-to-peer grids,” in *International Conference on Middleware*, Rio de Janeiro, Brazil, 2003, pp. 41–61.

[69] G. Fox, S. Pallickara, M. Pierce, and H. Gadgil, “Building messaging substrates for web and grid applications,” *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 363, no. 1833, pp. 1757–1773, 2005.

[70] G. Fox and S. Pallickara, “Deploying the NaradaBrokering substrate in aiding efficient web and grid service interactions,” *Proceedings of the IEEE*, vol. 93, no. 3, pp. 564–577, Mar. 2005.

[71] S. Pallickara and G. Fox, “On the matching of events in distributed brokering systems,” in *IEEE ITCC Conference on Information Technology*, 2004, vol. 2, pp. 68–76.

[72] S. Pallickara and G. Fox, “A scheme for reliable delivery of events in distributed middleware systems,” in *1st International Conference on Autonomic Computing*, New York, United States, 2004, pp. 328–329.

[74] H. Bulut, S. Pallickara, and G. Fox, “Implementing a NTP-based time service within a distributed middleware system,” in *3rd International Symposium on Principles and Practice of Programming in Java*, Las Vegas, Nevada, 2004, pp. 126–134.

[75] A. Fast, D. Jensen, and B. N. Levine, “Creating social networks to improve peer-to-peer networking,” in *11th ACM SIGKDD international conference on Knowledge discovery in data mining*, Chicago, Illinois, 2005, pp. 568–573.

[76] M. Al-Smadi and C. Gütl, “SOA-based architecture for a generic and flexible e-assessment system,” in *1st Annual Engineering Education Conference*, Madrid, Spain, 2010, pp. 493–500.

[77] S. Birrer and F. E. Bustamante, “Resilient peer-to-peer multicast from the ground up,” in *3rd IEEE International Symposium on Network Computing and Applications*, Cambridge, MA, USA, 2004, pp. 351–355.

[78] S. Birrer and F. E. Bustamante, “Resilience in overlay multicast protocols,” in *14th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, Monterey, California, 2006, pp. 363–372.

[79] S. Birrer, D. Lu, F. Bustamante, Y. Qiao, and P. Dinda, “FatNemo: Building a resilient multi-source multicast fat-tree,” *Web Content Caching and Distribution*, pp. 182–196, 2004.

[80] E. Cohen and S. Shenker, “Replication strategies in unstructured peer-to-peer networks,” *ACM SIGCOMM Computer Communication Review*, vol. 32, pp. 177–190, 2002.

[81] K. Koh, S. Han, and E. Kim, “Efficient load balancing system for overlay multicast,” in *The International Conference on Information Networking 2008*, Busan, Korea, 2008, pp. 1–4.

[84] S. Banerjee, Seungjoon Lee, B. Bhattacharjee, and A. Srinivasan, “Resilient multicast using overlays,” *IEEE/ACM Transactions on Networking*, vol. 14, no. 2, pp. 237–248, 2006.

[87] M. Piatek, C. Dixon, A. Krishnamurthy, and T. Anderson, “LiveSwarms: adapting BitTorrent for end host multicast,” University of Washington, Technical TR 2006-11-01, 2006.

[88] M. Waldvogel and R. Rinaldi, “Efficient topology-aware overlay network,” *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 1, pp. 101–106, 2003.

[89] J. Liebeherr, M. Nahas, and W. Si, “Large-scale application-layer multicast with delaunay triangulations,” in *IEEE Global Telecommunications Conference*, San Antonio, Texas, 2001.

[90] A. Rowstron and P. Druschel, “Pastry: scalable, decentralized object location, and routing for large-scale peer-to-peer systems,” in *IFIP/ACM International Conference on Distributed Systems Platforms*, Heidelberg, Germany, 2001, pp. 329–350.

[91] M. Castro, P. Druschel, A. M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, “SplitStream: high-bandwidth multicast in cooperative environments,” in *19th ACM Symposium on Operating Systems Principles*, 2003, vol. 37, pp. 298–313.

[92] S. Birrer and F. E. Bustamante, “Magellan: Performance-based, cooperative multicast,” in *10th International Workshop on Web Content Caching and Distribution*, French Riviera, France, 2005, pp. 133–143.

[93] X. Li, B. YAN, and W. LUO, “Overlay multicast network optimization and simulation based on Narada protocol,” in *10th International Conference on Advanced Communication Technology*, ICACT 2008, 2008, pp. 2215–2220.

[94] T. Steele, V. Vishnumurthy, and P. Francis, “A parameter-free load balancing mechanism for p2p networks,” in *7th International Conference on Peer-to-Peer Systems*, Berkeley, California, 2008.

[95] V. Venkataraman, K. Yoshida, and P. Francis, “Chunkyspread: heterogeneous unstructured tree-based peer-to-peer multicast,” in *14th IEEE International Conference on Network Protocols*, Santa Barbara, California, 2006, pp. 2–11.

[96] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron, “SCRIBE: a large-scale and decentralized application-level multicast infrastructure,” *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 8, pp. 100–110, 2002.

[97] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, “Application-level multicast using content-addressable networks,” *Networked Group Communication*, pp. 14–29, 2001.

[98] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz, “Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination,” in *11th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, New York, United States, 2001, pp. 11–20.

[99] S. Birrer, “Bandwidth intensive application-layer multicast in dynamic environments,” in *IEEE Infocom 2005 Conference*, Miami, Florida, 2005.

[100] Z. Wang and J. Crowcroft, “Bandwidth-delay based routing algorithms,” in *IEEE Global Telecommunications Conference*, Singapore, 1995, vol. 3, pp. 2129–2133.

[101] D. Lu and P. A. Dinda, “Synthesizing realistic computational grids,” in *Conference on High Performance Computing Networking, Storage and Analysis*, Phoenix, Arizona, 2003, p. 16.

[102] D. M. Moen, “Overview of overlay multicast protocols,” *C3I Center, George Mason University*, 2004.

[104] S. Birrer and F. E. Bustamante, “Resilient peer-to-peer multicast without the cost,” in *12th Annual Multimedia Computing and Networking Conference*, San Jose, California, 2005, vol. 5680, pp. 113–120.

[106] V. N. Padmanabhan, H. J. Wang, and P. A. Chou, “Resilient peer-to-peer streaming,” in *11th IEEE International Conference on Network Protocols*, Atlanta, Georgia, 2003, pp. 16–27.

[107] K. F. S. Wong, S. H. G. Chan, W. C. Wong, Q. Zhang, W. W. Zhu, and Y. Q. Zhang, “Lateral error recovery for application-level multicast,” in *23rd AnnualJoint Conference of the IEEE Computer and Communications Societies*, Hong Kong, 2004, vol. 4, pp. 2708–2718.

[108] Z. Unal and A. Unal, “Evaluating and comparing the usability of web-based course management systems,” *Journal of Information Technology Education*, vol. 10, pp. 19–38, 2011.

[109] L. A. M. Zaina, G. Bressan, R. M. Silveira, I. Stiubiener, and W. V. Ruggiero, “Analysis and comparison of distance education environments,” in *International Conference on Engineering Education*, Oslo, Norway, 2001.

[110] “Blackboard buys competitor for $180 M,” *Washington Business Journal*, Washington, DC, 12-Oct-2005.

[111] “Moodle statistics as of November 2011,” *Moodle*. [Online]. Available: http://moodle.org/stats. [Accessed: 05-Nov-2011].

[112] S. Schullo, A. Hilbelink, M. Venable, and A. Barron, “Selecting a virtual classroom system: Elluminate Live vs. Macromedia Breeze,” *MERLOT Journal of Online Learning and Teaching*, vol. 3, no. 4, pp. 331–345, 2007.

[113] E. Lavolette, M. Venable, E. Gose, and P. Huang, “Comparing synchronous virtual classrooms: student, instructor and course designer perspectives,” *Tech Trends*, vol. 54, no. 5, pp. 54–61, 2010.

[114] “About Bb.” [Online]. Available: http://www.blackboard.com/About-Bb/Our-Story.aspx. [Accessed: 14-Apr-2013].

[115] V. Roesler, F. Cecagno, L. Daronco, and F. Dixon, “Mconf: An open source multiconference system for web and mobile devices,” in *Multimedia- A Multidisciplinary Approach to Complex Issues*, InTech, 2012, pp. 203–228.

[116] “BigBlueButton Frequently Asked Questions,” *bigbluebutton*. [Online]. Available: http://code.google.com/p/bigbluebutton/wiki/FAQ#What_are_the_minimum_bandwidth_requirements_for_a_user? [Accessed: 14-Apr-2013].

[117] "Concepts and planning," *IBM*. [Online]. Available: http://publib.boulder.ibm.com/infocenter/txformp/v6r0m0/index.jsp?topic=%2Fcom.ibm.cics.te.doc%2Ferziaz0015.htm. [Accessed: 29-Feb-2012].

[118] A. Tanenbaum, *Computer Networks*, 4th ed. Prentice Hall, 2003.

[119] X. Jin, W. Tu, and S. H. G. Chan, "Challenges and advances in using IP multicast for overlay data delivery," *IEEE Communications Magazine*, vol. 47, no. 6, pp. 157–163, 2009.

[120] M. Srivastava, "Comparative study of current trends in distance education in Canada and India," *Turkish online Journal of Distance Education*, vol. 3, no. 4, 2002.

[121] "Smartphone savvy," 2012. [Online]. Available: http://zboostyourlife.wi-exblog.com/category/holiday-fun/page/2/. [Accessed: 09-Apr-2013].

[122] M. Piatek, T. Isdal, A. Krishnamurthy, and A. Venkataramani, "Do incentives build robustness in BitTorrent?," in *4th USENIX Symposium on Networked Systems Design & Implementation*, Cambridge, MA, USA, 2007.

[123] H. Zhang, L. Tang, and J. Li, "Impact of overlay routing on end-to-end delay," in *15th International Conference on Computer Communications and Networks*, Arlington, Virginia, 2006, pp. 435–440.

[125] "Performance overview," *Oracle*. [Online]. Available: http://docs.oracle.com/html/A90444_01/concpts.htm. [Accessed: 10-Jul-2012].

[126] A. Montresor and M. Jelasity, "PeerSim: A scalable P2P simulator," in *IEEE 9th International Conference on P2P*, Seattle, Washington, 2009, pp. 99–100.

[127] F. Frioli and M. Pedrolli, "A BitTorrent module for peersim." Feb-2008.

[128] N. Ahmed, T. Ahmed, and M. Faizullah, "Working of upazila parishad in Bangladesh," UNDP Bangladesh, 2010.

[129] "Mobile marvels: A special report on telecoms in emerging markets.," *The Economist*, 26-Sep-2009.

APPENDIX A: CONGESTION WINDOW SIMULATION CODE

This program (ns3) was used to analyze the TCP congestion window. The link speed and delay parameter values were changed in the code. The different values of number of packet and node parameters were entered on the command line.

Command to run the script is:

```
./waf --run "name --nCsma=  --nPackets= " > outputfilename.dat  2>&1
```

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 */

#include <fstream>
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/ipv4-global-routing-helper.h"
#include "ns3/csma-module.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("TCPCongestion");

class MyApp : public Application
{
public:

  MyApp ();
  virtual ~MyApp();

  void Setup (Ptr<Socket> socket, Address address, uint32_t packetSize, uint32_t
nPackets, DataRate dataRate);

private:
  virtual void StartApplication (void);
  virtual void StopApplication (void);

  void ScheduleTx (void);
  void SendPacket (void);
  Ptr<Socket>     m_socket;
  Address         m_peer;
  uint32_t        m_packetSize;
  uint32_t        m_nPackets;
  DataRate        m_dataRate;
  EventId         m_sendEvent;
  bool            m_running;
  uint32_t        m_packetsSent;
};

MyApp::MyApp ()
  : m_socket (0),
    m_peer (),
    m_packetSize (0),
    m_nPackets (0),
    m_dataRate (0),
    m_sendEvent (),
    m_running (false),
    m_packetsSent (0)
{
}

MyApp::~MyApp()
{
  m_socket = 0;
}

void
MyApp::Setup (Ptr<Socket> socket, Address address, uint32_t packetSize, uint32_t
nPackets, DataRate dataRate)
{
  m_socket = socket;
  m_peer = address;
  m_packetSize = packetSize;
  m_nPackets = nPackets;
  m_dataRate = dataRate;
}

void
MyApp::StartApplication (void)
{
  m_running = true;
  m_packetsSent = 0;
  m_socket->Bind ();
  m_socket->Connect (m_peer);
  SendPacket ();
}

void
MyApp::StopApplication (void)
{
  m_running = false;

  if (m_sendEvent.IsRunning ())
    {
      Simulator::Cancel (m_sendEvent);
    }

  if (m_socket)
    {
      m_socket->Close ();
    }
}

void
MyApp::SendPacket (void)
{
  Ptr<Packet> packet = Create<Packet> (m_packetSize);
  m_socket->Send (packet);

  if (++m_packetsSent < m_nPackets)
    {
      ScheduleTx ();
    }
}
```

```
void
MyApp::ScheduleTx (void)
{
  if (m_running)
    {
      Time tNext (Seconds (m_packetSize * 8 / static_cast<double>  (m_dataRate.GetBitRate
       ())));
      m_sendEvent = Simulator::Schedule (tNext, &MyApp::SendPacket, this);
    }
}

static void
CwndChange (uint32_t oldCwnd, uint32_t newCwnd)
{
  NS_LOG_UNCOND (Simulator::Now ().GetSeconds () << "\t" << newCwnd);
}

static void
RxDrop (Ptr<const Packet> p)
{
  NS_LOG_UNCOND ("RxDrop at " << Simulator::Now ().GetSeconds ());
}

int
main (int argc, char *argv[])
{
  uint32_t nCsma = 3;

  CommandLine cmd;
  cmd.AddValue ("nCsma", "Number of \"extra\" CSMA nodes/devices", nCsma);

  nCsma = nCsma == 0 ? 1 : nCsma;

  NodeContainer nodes;
  nodes.Create (2);

  NodeContainer csmaNodes;
  csmaNodes.Add (nodes.Get (1));
  csmaNodes.Create (nCsma);

  PointToPointHelper pointToPoint;

  pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("1Mbps"));
  pointToPoint.SetChannelAttribute ("Delay", StringValue ("5ms"));

  NetDeviceContainer p2pdevices;
  p2pdevices = pointToPoint.Install (nodes);
  CsmaHelper csma;
  csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
  csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));

  NetDeviceContainer csmaDevices;
  csmaDevices = csma.Install (csmaNodes);


// Ptr<RateErrorModel> em = CreateObjectWithAttributes<RateErrorModel> (
  //    "RanVar", RandomVariableValue (UniformVariable (0., 1.)),
    // "ErrorRate", DoubleValue (0.00001));
 // p2pdevices.Get (1)->SetAttribute ("ReceiveErrorModel", PointerValue (em));

  InternetStackHelper stack;
  stack.Install (nodes.Get (0));
  stack.Install (csmaNodes);


  Ipv4AddressHelper address;
  address.SetBase ("10.1.1.0", "255.255.255.0");

Ipv4InterfaceContainer p2pInterfaces;
  p2pInterfaces = address.Assign (p2pdevices);

  address.SetBase ("10.1.2.0", "255.255.255.0");
  Ipv4InterfaceContainer csmaInterfaces;
  csmaInterfaces = address.Assign (csmaDevices);
  uint16_t sinkPort = 8080;
  Address sinkAddress (InetSocketAddress (csmaInterfaces.GetAddress (nCsma), sinkPort));
  PacketSinkHelper packetSinkHelper ("ns3::TcpSocketFactory", InetSocketAddress
(Ipv4Address::GetAny (), sinkPort));
  ApplicationContainer sinkApps = packetSinkHelper.Install (csmaNodes.Get (nCsma));
  sinkApps.Start (Seconds (0.));
  sinkApps.Stop (Seconds (150.));

Ptr<Socket> ns3TcpSocket = Socket::CreateSocket (nodes.Get (0),
TcpSocketFactory::GetTypeId ());
  ns3TcpSocket->TraceConnectWithoutContext ("CongestionWindow", MakeCallback
(&CwndChange));

  Ptr<MyApp> app = CreateObject<MyApp> ();
  app->Setup (ns3TcpSocket, sinkAddress, 1024, 128, DataRate ("1Mbps"));
  nodes.Get (0)->AddApplication (app);
  app->SetStartTime (Seconds (1.));
  app->SetStopTime (Seconds (150.));

  Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

 csmaDevices.Get (nCsma)->TraceConnectWithoutContext ("PhyRxDrop", MakeCallback
(&RxDrop));
  Simulator::Stop (Seconds (150));
  Simulator::Run ();
  Simulator::Destroy ();

  return 0;
}
```

APPENDIX B: P2P SIMULATION CODE

This main program (PeerSim) was used for P2P simulations. The fragment size parameter values were set in the code.

```
/*
 * Copyright (c) 2007-2008 Fabrizio Frioli, Michele Pedrolli
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU Lesser General Public License version 2 as
 * published by the Free Software Foundation.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

package peersim.bittorrent;

import peersim.core.*;
import peersim.config.*;
import peersim.edsim.*;
import peersim.transport.*;
import peersim.graph.Graph;
/**
 *     This is the class that implements the BitTorrent module for Peersim
 */
public class BitTorrent implements EDProtocol
{
       /**
        *     The size in Megabytes of the file being shared.
        *     @config
        */
  private static final String PAR_SIZE = "file_size";
       /**
        *     The Transport used by the the protocol.
        *     @config
        */
  private static final String PAR_TRANSPORT = "transport";
       /**
        *     The maximum number of neighbor that a node can have.
        *     @config
        */
  private static final String PAR_SWARM = "max_swarm_size";
       /**
        *     The maximum number of peers returned by the tracker when a new
        *     set of peers is requested through a <tt>TRACKER</tt> message.
        *     @config
        */
  private static final String PAR_PEERSET_SIZE = "peerset_size";
       /**
        *     Defines how much the network can grow with respect to the
<tt>network.size</tt>
        *  when {@link NetworkDynamics} is used.
        *     @config
```

```
         */
private static final String PAR_MAX_GROWTH = "max_growth";
        /**
         *      Is the number of requests of the same block sent to different peers.
         *      @config
         */
private static final String PAR_DUP_REQ = "duplicated_requests";

        /**
         *      KEEP_ALIVE message.
         *      @see SimpleEvent#type "Event types"
         */
private static final int KEEP_ALIVE = 1;

        /**
         *      CHOKE message.
         *      @see SimpleEvent#type "Event types"
         */
private static final int CHOKE = 2;

        /**
         *      UNCHOKE message.
         *      @see SimpleEvent#type "Event types"
         */
private static final int UNCHOKE = 3;

        /**
         *      INTERESTED message.
         *      @see SimpleEvent#type "Event types"
         */
private static final int INTERESTED = 4;

        /**
         *      NOT_INTERESTED message.
         *      @see SimpleEvent#type "Event types"
         */
private static final int NOT_INTERESTED = 5;

        /**
         *      HAVE message.
         *      @see SimpleEvent#type "Event types"
         */
private static final int HAVE = 6;

        /**
         *      BITFIELD message.
         *      @see SimpleEvent#type "Event types"
         */
private static final int BITFIELD = 7;

        /**
         *      REQUEST message.
         *      @see SimpleEvent#type "Event types"
         */
private static final int REQUEST = 8;

        /**
         *      PIECE message.
         *      @see SimpleEvent#type "Event types"
         */
private static final int PIECE = 9;

        /**
         *      CANCEL message.
         *      @see SimpleEvent#type "Event types"
         */
private static final int CANCEL = 10;

        /**
         *      TRACKER message.
         *      @see SimpleEvent#type "Event types"
         */
private static final int TRACKER = 11;

        /**
         *      PEERSET message.
        *     @see SimpleEvent#type "Event types"
        */
  private static final int PEERSET = 12;

       /**
        *     CHOKE_TIME event.
        *     @see SimpleEvent#type "Event types"
        */
  private static final int CHOKE_TIME = 13;

       /**
        *     OPTUNCHK_TIME event.
        *     @see SimpleEvent#type "Event types"
        */
  private static final int OPTUNCHK_TIME = 14;

       /**
        *     ANTISNUB_TIME event.
        *     @see SimpleEvent#type "Event types"
        */
  private static final int ANTISNUB_TIME = 15;

       /**
        *     CHECKALIVE_TIME event.
        *     @see SimpleEvent#type "Event types"
        */
  private static final int CHECKALIVE_TIME = 16;

       /**
        *     TRACKERALIVE_TIME event.
        *     @see SimpleEvent#type "Event types"
        */
  private static final int TRACKERALIVE_TIME = 17;

       /**
        *     DOWNLOAD_COMPLETED event.
        *     @see SimpleEvent#type "Event types"
        */
  private static final int DOWNLOAD_COMPLETED = 18;

       /**
        *     The maximum connection speed of the local node.
        */
  int maxBandwidth;

       /**
        *     Stores the neighbors ordered by ID.
        *  @see Element
        */
  private peersim.bittorrent.Element byPeer[];

       /**
        *     Contains the neighbors ordered by bandwidth as needed by the unchoking
        *     algorithm.
        */
  private peersim.bittorrent.Element byBandwidth[];

       /**
        *     The Neighbors list.
        */
  private Neighbor cache[];

       /**
        *     Reference to the neighbors that unchoked the local node.
        */
  private boolean unchokedBy[];

       /**
        *     Number of neighbors in the cache. When it decreases under 20, a new
peerset
        *     is requested to the tracker.
        */
  private int nNodes = 0;

       /**
        *     Maximum number of nodes in the network.
```

```
         */
  private int nMaxNodes;

        /**
         *     The status of the local peer. 0 means that the current peer is a leecher,
1 a seeder.
         */
  private int peerStatus;

        /**
         *     Defines how much the network can grow with respect to the
<tt>network.size</tt>
         *  when {@link NetworkDynamics} is used.
         */
  public int maxGrowth;

        /**
         *     File status of the local node. Contains the blocks owned by the local
node.
         */
  private int status[];

        /**
         *     Current number of Bitfield request sent. It must be taken into account
         *     before sending another one.
         */
  private int nBitfieldSent = 0;

        /**
         *     Current number of pieces in upload from the local peer.
         */
  public int nPiecesUp = 0;
        /**
         *     Current number of pieces in download to the local peer.
         */
  public int nPiecesDown = 0;

        /**
         *     Current number of piece completed.
         */
  private int nPieceCompleted = 0;

        /**
         *     Current downloading piece ID, the previous lastInterested piece.
         */
  int currentPiece = -1;

        /**
         *     Used to compute the average download rates in choking algorithm. Stores
the
         *     number of <tt>CHOKE</tt> events.
         */
  int n_choke_time = 0;

        /**
         *     Used to send the <tt>TRACKER</tt> message when the local node has 20
neighbors
         *     for the first time.
         */
  boolean lock = false;

        /**
         *     Number of peers interested in my pieces.
         */
  int numInterestedPeers = 0;

        /**
         *     Last piece for which the local node sent an <tt>INTERESTED</tt> message.
         */
  int lastInterested = -1;

        /**
         *     The status of the current piece in download. Length 16, every time the
local node
         *     receives a PIECE message, it updates the corresponding block's cell. This
cell
```

```
         *      contain the ID for that block of that piece. If
         *      block already received this is discarded.
         */
  private int pieceStatus[];

        /**
         *      Length of the file. Stored as number of pieces (256KB each one).
         */
  int nPieces;

        /**
         *      Contains the neighbor's status of the file. Every row represents a
         *      node and every cell has value 0 if the neighbor doesn't
         *      have the piece, 1 otherwise. It has {@link #swarmSize} rows and {@link
#nPieces}
         *      columns.
         */
  int[][] swarm;

        /**
         *      The summation of the swarm's rows. Calculated every time a {@link
#BITFIELD} message
         *      is received and updated every time HAVE message is received.
         */
  int rarestPieceSet[];

        /**
         *      The five pending block requests.
         */
  int pendingRequest[];

        /**
         *      The maximum swarm size (default is 80)
         */
  int swarmSize;

        /**
         *      The size of the peerset. This is the number of "friends" nodes
         *      sent from the tracker to each new node (default: 50)
         */
  int peersetSize;

        /**
         * The ID of the current node
         */
  private long thisNodeID;

        /**
     *  Number of duplicated requests as specified in the configuration file.
         *      @see BitTorrent#PAR_DUP_REQ
         */
  private int numberOfDuplicatedRequests;

        /**
         *      The queue where the requests to serve are stored.
         *      The default dimension of the queue is 20.
         */
  Queue requestToServe = null;

        /**
         *      The queue where the out of sequence incoming pieces are stored
         *      waiting for the right moment to be processed.
     *  The default dimension of the queue is 100.
         */
  Queue incomingPieces = null;

        /**
         *      The Transport ID.
         *      @see BitTorrent#PAR_TRANSPORT
         */
  int tid;

        /**
         *      The reference to the tracker node. If equals to <tt>null</tt>, the local
         *      node is the tracker.
         */
  private Node tracker = null;

        /**
         *     The default constructor. Reads the configuration file and initializes the
         *     configuration parameters.
         *     @param prefix the component prefix declared in the configuration file
         */
  public BitTorrent (String prefix)
  {                          // Used for the tracker's protocol
    tid = Configuration.getPid (prefix + "." + PAR_TRANSPORT);
// Here the total file size is obtained from the configuration file

// For 32 KB fragment size
    nPieces =
      (int) ((Configuration.getInt (prefix + "." + PAR_SIZE)) * 1024 / 512);

// For 64 KB fragment size
    nPieces =
      (int) ((Configuration.getInt (prefix + "." + PAR_SIZE)) * 1024 / 1024);

// For 128 KB fragment size
    nPieces =
      (int) ((Configuration.getInt (prefix + "." + PAR_SIZE)) * 1024 / 2048);

    swarmSize = (int) Configuration.getInt (prefix + "." + PAR_SWARM);
    peersetSize =
      (int) Configuration.getInt (prefix + "." + PAR_PEERSET_SIZE);
    numberOfDuplicatedRequests =
      (int) Configuration.getInt (prefix + "." + PAR_DUP_REQ);
    maxGrowth = (int) Configuration.getInt (prefix + "." + PAR_MAX_GROWTH);
    nMaxNodes = Network.getCapacity () - 1;
  }

        /**
         *     Gets the reference to the tracker node.
         *     @return the reference to the tracker
         */
  public Node getTracker ()
  {
    return tracker;
  }

        /**
         *     Gets the number of neighbors currently stored in the cache of the local
node.
         *     @return the number of neighbors in the cache
         */
  public int getNNodes ()
  {
    return this.nNodes;
  }

        /**
         *     Sets the reference to the tracker node.
         *     @param t the tracker node
         */
  public void setTracker (Node t)
  {
    tracker = t;
  }

        /**
         *     Sets the ID of the local node.
         *     @param id the ID of the node
         */
  public void setThisNodeID (long id)
  {
    this.thisNodeID = id;
  }

        /**
         *     Gets the ID of the local node.
         *     @return the ID of the local node
         */
  public long getThisNodeID ()
  {
    return this.thisNodeID;
  }

        /**
         *     Gets the file status of the local node.
         *     @return the file status of the local node
         */
  public int[] getFileStatus ()
  {
    return this.status;
  }

        /**
         *     Initializes the tracker node. This method
         *     only performs the initialization of the tracker's cache.
         */
  public void initializeTracker ()
  {
    cache = new Neighbor[nMaxNodes + maxGrowth];
    for (int i = 0; i < nMaxNodes + maxGrowth; i++)
      {
        cache[i] = new Neighbor ();
      }
  }

        /**
         *     <p>Checks the number of neighbors and if it is equal to 20
         *     sends a TRACKER messages to the tracker, asking for a new
         *     peer set.</p>
         *
         *     <p>This method *must* be called after every call of {@link
#removeNeighbor}
         *     in {@link #processEvent}.
         *     </p>
         */
  private void processNeighborListSize (Node node, int pid)
  {
    if (nNodes == 20)
      {
        Object ev;
        long latency;
        ev = new SimpleMsg (TRACKER, node);
        Node tracker = ((BitTorrent) node.getProtocol (pid)).tracker;
        if (tracker != null)
          {
            latency =
              ((Transport) node.getProtocol (tid)).getLatency (node, tracker);
            EDSimulator.add (latency, ev, tracker, pid);
          }
      }
  }

        /**
         *     The standard method that processes incoming events.
         *     @param node reference to the local node for which the event is going to be
processed
         *     @param pid BitTorrent's protocol id
         *     @param event the event to process
         */
  public void processEvent (Node node, int pid, Object event)
  {

    Object ev;
    long latency;
    switch (((SimpleEvent) event).getType ())
      {

      case KEEP_ALIVE:        // 1
        {
          Node sender = ((IntMsg) event).getSender ();
          int isResponse = ((IntMsg) event).getInt ();
          //System.out.println("process, keep_alive: sender is "+sender.getID()+", local
is "+node.getID());
          Element e = search (sender.getID ());
          if (e != null)
            {                 //if I know the sender
```

```
              cache[e.peer].isAlive ();
              if (isResponse == 0 && alive (sender))
                {
                  Object msg = new IntMsg (KEEP_ALIVE, node, 1);
                  latency =
                    ((Transport) node.getProtocol (tid)).getLatency (node,
                                                                     sender);
                  EDSimulator.add (latency, msg, sender, pid);
                  cache[e.peer].justSent ();
                }
            }
          else
            {
              System.err.
                println ("despite it should never happen, it happened");
              ev =
                new BitfieldMsg (BITFIELD, true, false, node, status,
                                 nPieces);
              latency =
                ((Transport) node.getProtocol (tid)).getLatency (node,
                                                                 sender);
              EDSimulator.add (latency, ev, sender, pid);
              nBitfieldSent++;
            }

        };
        break;

      case CHOKE:             // 2, CHOKE message.
        {
          Node sender = ((SimpleMsg) event).getSender ();
          //System.out.println("process, choke: sender is "+sender.getID()+", local is
"+node.getID());
          Element e = search (sender.getID ());
          if (e != null)
            {                 //if I know the sender
              cache[e.peer].isAlive ();
              unchokedBy[e.peer] = false;    // I'm choked by it
            }
          else
            {
              System.err.
                println ("despite it should never happen, it happened");
              ev =
                new BitfieldMsg (BITFIELD, true, false, node, status,
                                 nPieces);
              latency =
                ((Transport) node.getProtocol (tid)).getLatency (node,
                                                                 sender);
              EDSimulator.add (latency, ev, sender, pid);
              nBitfieldSent++;
            }
        };
        break;

      case UNCHOKE:           // 3, UNCHOKE message.
        {



          Node sender = ((SimpleMsg) event).getSender ();
          //System.out.println("process, unchoke: sender is "+sender.getID()+", local is
"+node.getID());
          Element e = search (sender.getID ());
          if (e != null)
            {                 // If I know the sender
              int senderIndex = e.peer;
              cache[senderIndex].isAlive ();
              /* I send to it some of the pending requests not yet satisfied. */
              int t = numberOfDuplicatedRequests;
              for (int i = 4; i >= 0 && t > 0; i--)
                {
                  if (pendingRequest[i] == -1)
                    break;
                  if (alive (cache[senderIndex].node)
                      && swarm[senderIndex][decode (pendingRequest[i], 0)] ==
                          1)
                        {          //If the sender has that piece
                          ev = new IntMsg (REQUEST, node, pendingRequest[i]);
                          latency =
                            ((Transport) node.getProtocol (tid)).getLatency (node,
                                                                    sender);
                          EDSimulator.add (latency, ev, sender, pid);
                          cache[senderIndex].justSent ();
                        }
                      if (!alive (cache[senderIndex].node))
                        {
                          System.out.println ("unchoke1 rm neigh " +
                                              cache[i].node.getID ());
                          removeNeighbor (cache[senderIndex].node);
                          processNeighborListSize (node, pid);
                          return;
                        }
                      t--;
                    }
                  // I request missing blocks to fill the queue
                  int block = getBlock ();
                  int piece;
                  while (block != -2)
                    {              //while still available request to send
                      if (block < 0)
                        {          // No more block to request for the current piece
                          piece = getPiece ();
                          if (piece == -1)
                            {      // no more piece to request
                              break;
                            }
                          for (int j = 0; j < swarmSize; j++)
                            {      // send the interested message to those
                              // nodes which have that piece
                              lastInterested = piece;
                              if (alive (cache[j].node) && swarm[j][piece] == 1)
                                {

                                  ev =
                                    new IntMsg (INTERESTED, node, lastInterested);
                                  latency =
                                    ((Transport) node.getProtocol (tid)).
                                    getLatency (node, cache[j].node);
                                  EDSimulator.add (latency, ev, cache[j].node,
                                                   pid);
                                  cache[j].justSent ();
                                }

                              if (!alive (cache[j].node))
                                {
                                  //System.out.println("unchoke2 rm neigh "+
cache[j].node.getID() );
                                  removeNeighbor (cache[j].node);
                                  processNeighborListSize (node, pid);
                                }
                            }
                          block = getBlock ();
                        }
                      else
                        {          // block value referred to a real block
                          if (alive (cache[senderIndex].node)
                              && swarm[senderIndex][decode (block, 0)] == 1
                              && addRequest (block))
                            {      // The sender has that block
                              ev = new IntMsg (REQUEST, node, block);
                              latency =
                                ((Transport) node.getProtocol (tid)).
                                getLatency (node, sender);
                              EDSimulator.add (latency, ev, sender, pid);
                              cache[senderIndex].justSent ();
                            }
                          else
                            {
                              if (!alive (cache[senderIndex].node))
                                {
                                  System.out.println ("unchoke3 rm neigh " +
                                             cache[senderIndex].node.
                                             getID ());
                                  removeNeighbor (cache[senderIndex].node);
                                  processNeighborListSize (node, pid);
                                }
                              return;
                            }
                            block = getBlock ();
                        }
                    }
                  unchokedBy[senderIndex] = true; // I add the sender to the list
                }
              else                  // It should never happen.
                {
                  System.err.
                    println ("despite it should never happen, it happened");
                  for (int i = 0; i < swarmSize; i++)
                    if (cache[i].node != null)
                      System.err.println (cache[i].node.getID ());
                  ev =
                    new BitfieldMsg (BITFIELD, true, false, node, status,
                                     nPieces);
                  latency =
                    ((Transport) node.getProtocol (tid)).getLatency (node,
                                                                     sender);
                  EDSimulator.add (latency, ev, sender, pid);
                  nBitfieldSent++;
                }
            };
            break;

          case INTERESTED:          // 4, INTERESTED message.
            {
              numInterestedPeers++;
              Node sender = ((IntMsg) event).getSender ();
              //System.out.println("process, interested: sender is "+sender.getID()+", local
is "+node.getID());
              int value = ((IntMsg) event).getInt ();
              Element e = search (sender.getID ());
              if (e != null)
                {
                  cache[e.peer].isAlive ();
                  cache[e.peer].interested = value;
                }
              else
                {
                  System.err.
                    println ("despite it should never happen, it happened");
                  ev =
                    new BitfieldMsg (BITFIELD, true, false, node, status,
                                     nPieces);
                  latency =
                    ((Transport) node.getProtocol (tid)).getLatency (node,
                                                                     sender);
                  EDSimulator.add (latency, ev, sender, pid);
                  nBitfieldSent++;
                }

            };
            break;

          case NOT_INTERESTED:      // 5, NOT_INTERESTED message.
            {
              numInterestedPeers--;
              Node sender = ((IntMsg) event).getSender ();
              //System.out.println("process, not_interested: sender is "+sender.getID()+",
local is "+node.getID());
              int value = ((IntMsg) event).getInt ();
              Element e = search (sender.getID ());
              if (e != null)
                {
                  cache[e.peer].isAlive ();
                  if (cache[e.peer].interested == value)
                    cache[e.peer].interested = -1;       // not interested
                }
            };
```

```
        break;

      case HAVE:              // 6, HAVE message.
        {


          Node sender = ((IntMsg) event).getSender ();
          //System.out.println("process, have: sender is "+sender.getID()+", local is
"+node.getID());
          int piece = ((IntMsg) event).getInt ();
          Element e = search (sender.getID ());
          if (e != null)
            {
              cache[e.peer].isAlive ();
              swarm[e.peer][piece] = 1;
              rarestPieceSet[piece]++;
              boolean isSeeder = true;
              for (int i = 0; i < nPieces; i++)
                {
                  isSeeder = isSeeder && (swarm[e.peer][i] == 1);
                }
              e.isSeeder = isSeeder;
            }
          else
            {
              System.err.
                println ("despite it should never happen, it happened");
              ev =
                new BitfieldMsg (BITFIELD, true, false, node, status,
                                 nPieces);
              latency =
                ((Transport) node.getProtocol (tid)).getLatency (node,
                                                                 sender);
              EDSimulator.add (latency, ev, sender, pid);
              nBitfieldSent++;
            }
        };
        break;

      case BITFIELD:          // 7, BITFIELD message
        {


          Node sender = ((BitfieldMsg) event).getSender ();
          int[] fileStatus = ((BitfieldMsg) event).getArray ();
          /*Response with NACK */
          if (!((BitfieldMsg) event).isRequest && !((BitfieldMsg) event).ack)
            {
              Element e = search (sender.getID ());
              if (e == null)  // if is a response with nack that follows a request
                nBitfieldSent--;
              // otherwise is a response with ack that follows a duplicate
              // insertion attempt
              //System.out.println("process, bitfield_resp_nack: sender is
"+sender.getID()+", local is "+node.getID());
              return;
            }
          /*Request with NACK */
          if (((BitfieldMsg) event).isRequest && !((BitfieldMsg) event).ack)
            {
              //System.out.println("process, bitfield_req_nack: sender is
"+sender.getID()+", local is "+node.getID());
              if (alive (sender))
                {
                  Element e = search (sender.getID ());
                  ev = new BitfieldMsg (BITFIELD, false, true, node, status, nPieces);
        //response with ack
                  latency =
                    ((Transport) node.getProtocol (tid)).getLatency (node,
                                                                     sender);
                  EDSimulator.add (latency, ev, sender, pid);
                  cache[e.peer].justSent ();
                }
            }
          /*Response with ACK */
          if (!((BitfieldMsg) event).isRequest && ((BitfieldMsg) event).ack)
```

```
            {
              nBitfieldSent--;
              //System.out.println("process, bitfield_resp_ack: sender is
"+sender.getID()+", local is "+node.getID());
              if (alive (sender))
                {
                  if (addNeighbor (sender))
                    {
                      Element e = search (sender.getID ());
                      cache[e.peer].isAlive ();
                      swarm[e.peer] = fileStatus;
                      boolean isSeeder = true;
                      for (int i = 0; i < nPieces; i++)
                        {
                          rarestPieceSet[i] += fileStatus[i];
                          isSeeder = isSeeder && (fileStatus[i] == 1);
                        }
                      e.isSeeder = isSeeder;

                      if (nNodes == 10 && !lock)
                        {      // I begin to request pieces
                          lock = true;
                          int piece = getPiece ();
                          if (piece == -1)
                            return;
                          lastInterested = piece;
                          currentPiece = lastInterested;
                          ev = new IntMsg (INTERESTED, node, lastInterested);
                          for (int i = 0; i < swarmSize; i++)
                            {  // send the interested message to those
                              // nodes which have that piece
                              if (alive (cache[i].node)
                                  && swarm[i][piece] == 1)
                                {

                                  latency =
                                    ((Transport) node.getProtocol (tid)).
                                    getLatency (node, cache[i].node);
                                  EDSimulator.add (latency, ev, cache[i].node,
                                                   pid);
                                  cache[i].justSent ();
                                }
                            }

                        }

                    }
                }
              else
                System.out.println ("Sender " + sender.getID () +
                                    " not alive");
            }
          /*Request with ACK */
          if (((BitfieldMsg) event).isRequest && ((BitfieldMsg) event).ack)
            {
              //System.out.println("process, bitfield_req_ack: sender is
"+sender.getID()+", local is "+node.getID());
              if (alive (sender))
                {
                  if (addNeighbor (sender))
                    {
                      Element e = search (sender.getID ());
                      cache[e.peer].isAlive ();
                      swarm[e.peer] = fileStatus;
                      boolean isSeeder = true;
                      for (int i = 0; i < nPieces; i++)
                        {
                          rarestPieceSet[i] += fileStatus[i]; // I update the
rarestPieceSet with the pieces of the new node
                          isSeeder = isSeeder && (fileStatus[i] == 1);      // I check if
the new node is a seeder
                        }
                      e.isSeeder = isSeeder;
                      ev = new BitfieldMsg (BITFIELD, false, true, node, status, nPieces);
        //response with ack
                      latency =
                    ((Transport) node.getProtocol (tid)).getLatency (node,
                                                               sender);
                  EDSimulator.add (latency, ev, sender, pid);
                  cache[e.peer].justSent ();
                  if (nNodes == 10 && !lock)
                    {       // I begin to request pieces
                      int piece = getPiece ();
                      if (piece == -1)
                        return;
                      lastInterested = piece;
                      currentPiece = lastInterested;
                      ev = new IntMsg (INTERESTED, node, lastInterested);
                      for (int i = 0; i < swarmSize; i++)
                        {   // send the interested message to those
                          // nodes which have that piece
                          if (alive (cache[i].node)
                              && swarm[i][piece] == 1)
                            {

                              latency =
                                ((Transport) node.getProtocol (tid)).
                                getLatency (node, cache[i].node);
                              EDSimulator.add (latency, ev, cache[i].node,
                                               pid);
                              cache[i].justSent ();
                            }
                        }

                    }
                }
              else
                {
                  Element e;
                  if ((e = search (sender.getID ())) != null)
                    {       // The sender was already in the cache
                      cache[e.peer].isAlive ();
                      ev = new BitfieldMsg (BITFIELD, false, true, node, status,
nPieces);      //response with ack
                      latency =
                        ((Transport) node.getProtocol (tid)).
                        getLatency (node, sender);
                      EDSimulator.add (latency, ev, sender, pid);
                      cache[e.peer].justSent ();
                    }
                  else
                    {       // Was not possible to add the sender (nBitfield+nNodes >
swarmSize)
                      ev = new BitfieldMsg (BITFIELD, false, false, node, status,
nPieces);      //response with nack
                      latency =
                        ((Transport) node.getProtocol (tid)).
                        getLatency (node, sender);
                      EDSimulator.add (latency, ev, sender, pid);
                    }
                }

            }
          else
            System.out.println ("Sender " + sender.getID () +
                                " not alive");
        }
      };
      break;

    case REQUEST:           // 8, REQUEST message.
      {
        Object evnt;
        Node sender = ((IntMsg) event).getSender ();
        int value = ((IntMsg) event).getInt ();
        Element e;
        BitTorrent senderP;
        int remoteRate;
        int localRate;
        int bandwidth;
        int downloadTime;
```

```
          e = search (sender.getID ());
          if (e == null)
            return;
          cache[e.peer].isAlive ();

          requestToServe.enqueue (value, sender);

          /*I serve the enqueued requests until 10 uploading pieces or an empty queue */
          while (!requestToServe.empty () && nPiecesUp < 10)
            {
              Request req = requestToServe.dequeue ();
              e = search (req.sender.getID ());
              if (e != null && alive (req.sender))
                {
                  ev = new IntMsg (PIECE, node, req.id);
                  nPiecesUp++;
                  e.valueUP++;
                  senderP = ((BitTorrent) req.sender.getProtocol (pid));
                  senderP.nPiecesDown++;
                  remoteRate =
                    senderP.maxBandwidth / (senderP.nPiecesUp +
                                           senderP.nPiecesDown);
                  localRate = maxBandwidth / (nPiecesUp + nPiecesDown);
                  bandwidth = Math.min (remoteRate, localRate);
                  downloadTime = ((16 * 8) / (bandwidth)) * 1000;   // in milliseconds
                  latency =
                    ((Transport) node.getProtocol (tid)).getLatency (node,
                                                                   req.
                                                                   sender);
                  EDSimulator.add (latency + downloadTime, ev, req.sender,
                                   pid);
                  cache[e.peer].justSent ();
                  /*I send to me an event to indicate that the download is complete.
                     This prevents that, when the receiver death occurs, my value
nPiecesUp
                     doesn't decrease. */
                  evnt = new SimpleMsg (DOWNLOAD_COMPLETED, req.sender);
                  EDSimulator.add (latency + downloadTime, evnt, node, pid);
                }
            }
        };
        break;

      case PIECE:             // 9, PIECE message.
        {
          Node sender = ((IntMsg) event).getSender ();
          /*      Set the correct value for the local uploading and remote
             downloading number of pieces */
          nPiecesDown--;

          if (peerStatus == 1) // To save CPU cycles
            return;
          //System.out.println("process, piece: sender is "+sender.getID()+", local is
"+node.getID());
          Element e = search (sender.getID ());

          if (e == null)
            {                 //I can't accept a piece not wait
              return;
            }
          e.valueDOWN++;

          cache[e.peer].isAlive ();

          int value = ((IntMsg) event).getInt ();
          int piece = decode (value, 0);
          int block = decode (value, 1);
          /* If the block has not been already downloaded and it belongs to
             the current downloading piece. */
          if (piece == currentPiece
              && decode (pieceStatus[block], 0) != piece)
            {
              pieceStatus[block] = value;
              status[piece]++;
              removeRequest (value);
```

```
                requestNextBlocks (node, pid, e.peer);

              }
            else
              {                 // Either a future piece or an owned piece
                if (piece != currentPiece && status[piece] != 16)
                  {             // Piece not owned, will be considered later
                    incomingPieces.enqueue (value, sender);
                  }

              }
            ev = new IntMsg (CANCEL, node, value);
            /* I send a CANCEL to all nodes to which I previously requested the block */
            for (int i = 0; i < swarmSize; i++)
              {
                if (alive (cache[i].node) && unchokedBy[i] == true
                    && swarm[i][decode (block, 0)] == 1
                    && cache[i].node != sender)
                  {
                    latency =
                      ((Transport) node.getProtocol (tid)).getLatency (node,
                                                                        cache[i].
                                                                        node);
                    EDSimulator.add (latency, ev, cache[i].node, pid);
                    cache[i].justSent ();
                  }
              }

            if (status[currentPiece] == 16)
              {                 // if piece completed, I change the currentPiece to the
next wanted
                nPieceCompleted++;
                ev = new IntMsg (HAVE, node, currentPiece);
                for (int i = 0; i < swarmSize; i++)
                  {             // I send the HAVE for the piece
                    if (alive (cache[i].node))
                      {
                        latency =
                          ((Transport) node.getProtocol (tid)).getLatency (node,
                                                                            cache
                                                                            [i].
                                                                            node);
                        EDSimulator.add (latency, ev, cache[i].node, pid);
                        cache[i].justSent ();
                      }
                    if (!alive (cache[i].node))
                      {
                        //System.out.println("piece3 rm neigh "+ cache[i].node.getID() );

                        removeNeighbor (cache[i].node);
                        processNeighborListSize (node, pid);
                      }
                  }
                ev = new IntMsg (NOT_INTERESTED, node, currentPiece);
                for (int i = 0; i < swarmSize; i++)
                  {             // I send the NOT_INTERESTED to the peer I sent an
INTERESTED
                    if (swarm[i][piece] == 1 && alive (cache[i].node))
                      {
                        latency =
                          ((Transport) node.getProtocol (tid)).getLatency (node,
                                                                            cache
                                                                            [i].
                                                                            node);
                        EDSimulator.add (latency, ev, cache[i].node, pid);
                        cache[i].justSent ();
                      }
                    if (!alive (cache[i].node))
                      {
                        //System.out.println("piece4 rm neigh "+ cache[i].node.getID() );

                        removeNeighbor (cache[i].node);
                        processNeighborListSize (node, pid);
                      }
                  }
                if (nPieceCompleted == nPieces)
                {
                  System.out.println ("FILE COMPLETED for peer " +
                                      node.getID ());
                  this.peerStatus = 1;
                }

              /*      I set the currentPiece to the lastInterested. Then I extract
                 the queued received blocks
               */

              currentPiece = lastInterested;
              int m = incomingPieces.dim;
              while (m > 0)
                {             // I process the queue
                  m--;
                  Request temp = incomingPieces.dequeue ();
                  int p = decode (temp.id, 0);       // piece id
                  int b = decode (temp.id, 1);       // block id
                  Element s = search (temp.sender.getID ());
                  if (s == null)      // if the node that sent the block in the queue is
dead
                    continue;
                  if (p == currentPiece && decode (pieceStatus[b], 0) != p)
                    {
                      pieceStatus[b] = temp.id;
                      status[p]++;
                      removeRequest (temp.id);
                      requestNextBlocks (node, pid, s.peer);
                    }
                  else
                    {         // The piece not currently desired will be moved to the
tail
                      if (p != currentPiece)  // If not a duplicate block but belongs to
another piece
                        incomingPieces.enqueue (temp.id, temp.sender);
                      else     // duplicate block
                        requestNextBlocks (node, pid, s.peer);
                    }
                }
            }
        };
        break;

      case CANCEL:
        {
          Node sender = ((IntMsg) event).getSender ();
          int value = ((IntMsg) event).getInt ();
          requestToServe.remove (sender, value);
        };
        break;

      case PEERSET:           // PEERSET message
        {
          Node sender = ((PeerSetMsg) event).getSender ();
          //System.out.println("process, peerset: sender is "+sender.getID()+", local is
"+node.getID());
          Neighbor n[] = ((PeerSetMsg) event).getPeerSet ();

          for (int i = 0; i < peersetSize; i++)
            {
              if (n[i] != null && alive (n[i].node)
                  && search (n[i].node.getID ()) == null
                  && nNodes + nBitfieldSent < swarmSize - 2)
                {
                  ev =
                    new BitfieldMsg (BITFIELD, true, true, node, status,
                                     nPieces);
                  latency =
                    ((Transport) node.getProtocol (tid)).getLatency (node,
                                                                     n[i].
                                                                     node);
                  EDSimulator.add (latency, ev, n[i].node, pid);
                  nBitfieldSent++;
                  // Here I should call the Neighbor.justSent(), but here
                  // the node is not yet in the cache.
                }
              }
        };
        break;

      case TRACKER:           // TRACKER message
        {

          int j = 0;
          Node sender = ((SimpleMsg) event).getSender ();
          //System.out.println("process, tracker: sender is "+sender.getID()+", local is
"+node.getID());
          if (!alive (sender))
            return;
          Neighbor tmp[] = new Neighbor[peersetSize];
          int k = 0;
          if (nNodes <= peersetSize)
              {
                for (int i = 0; i < nMaxNodes + maxGrowth; i++)
                {
                  if (cache[i].node != null
                      && cache[i].node.getID () != sender.getID ())
                    {
                      tmp[k] = cache[i];
                      k++;
                    }
                }
                ev = new PeerSetMsg (PEERSET, tmp, node);
                latency =
                ((Transport) node.getProtocol (tid)).getLatency (node,
                                                                 sender);
                EDSimulator.add (latency, ev, sender, pid);
                return;
              }

          while (j < peersetSize)
              {
              int i = CommonState.r.nextInt (nMaxNodes + maxGrowth);
              for (int z = 0; z < j; z++)
                {
                  if (cache[i].node == null
                      || tmp[z].node.getID () == cache[i].node.getID ()
                      || cache[i].node.getID () == sender.getID ())
                    {
                      z = 0;
                      i = CommonState.r.nextInt (nMaxNodes + maxGrowth);
                    }
                }
              if (cache[i].node != null)
                {
                  tmp[j] = cache[i];
                  j++;
                }
              }
          ev = new PeerSetMsg (PEERSET, tmp, node);
          latency =
              ((Transport) node.getProtocol (tid)).getLatency (node, sender);
          EDSimulator.add (latency, ev, sender, pid);
        };
        break;

      case CHOKE_TIME:        //Every 10 secs.
        {
          n_choke_time++;

          ev = new SimpleEvent (CHOKE_TIME);
          EDSimulator.add (10000, ev, node, pid);
          int j = 0;
          /*I copy the interested nodes in the byBandwidth array */
          for (int i = 0; i < swarmSize && byPeer[i].peer != -1; i++)
              {
              if (cache[byPeer[i].peer].interested > 0)
                {
                  byBandwidth[j] = byPeer[i]; //shallow copy
                  j++;
                }
              }
        /*It ensures that in the next 20sec, if there are less nodes interested
           than now, those in surplus will not be ordered. */
        for (; j < swarmSize; j++)
          {
            byBandwidth[j] = null;
          }
        sortByBandwidth ();
        int optimistic = 3;
        int luckies[] = new int[3];
        try
        {                   // It takes the first three neighbors
          luckies[0] = byBandwidth[0].peer;
          optimistic--;
          luckies[1] = byBandwidth[1].peer;
          optimistic--;
          luckies[2] = byBandwidth[2].peer;
        }
        catch (NullPointerException e)
        {                   // If not enough peer in byBandwidth it chooses the other
randomly
          for (int z = optimistic; z > 0; z--)
            {
              int lucky = CommonState.r.nextInt (nNodes);
              while (cache[byPeer[lucky].peer].status == 1 && alive
(cache[byPeer[lucky].peer].node) && cache[byPeer[lucky].peer].interested == 0)    //
until the lucky peer is already unchoked or not interested
                lucky = CommonState.r.nextInt (nNodes);
              luckies[3 - z] = byPeer[lucky].peer;
            }
        }
        for (int i = 0; i < swarmSize; i++)
          {                 // I perform the chokes and the unchokes
            if ((i == luckies[0] || i == luckies[1] || i == luckies[2])
                && alive (cache[i].node) && cache[i].status != 2)
              {             //the unchokes
                cache[i].status = 1;
                ev = new SimpleMsg (UNCHOKE, node);
                latency =
                  ((Transport) node.getProtocol (tid)).getLatency (node,
                                                                    cache[i].
                                                                    node);
                EDSimulator.add (latency, ev, cache[i].node, pid);
                cache[i].justSent ();
                //System.out.println("average time, unchoked: "+cache[i].node.getID());
              }
            else
              {             // the chokes
                if (alive (cache[i].node)
                    && (cache[i].status == 1 || cache[i].status == 2))
                  {
                    cache[i].status = 0;
                    ev = new SimpleMsg (CHOKE, node);
                    latency =
                      ((Transport) node.getProtocol (tid)).getLatency (node,
                                                                        cache
                                                                        [i].
                                                                        node);
                    EDSimulator.add (latency, ev, cache[i].node, pid);
                    cache[i].justSent ();
                  }
              }
          }

        if (n_choke_time % 2 == 0)
          {                 //every 20 secs. Used in computing the average download
rates
            for (int i = 0; i < nNodes; i++)
              {
                if (this.peerStatus == 0)
                  {         // I'm a leeacher
                    byPeer[i].head20 = byPeer[i].valueDOWN;
                  }
                else
                  {
                    byPeer[i].head20 = byPeer[i].valueUP;
                  }
              }
          }
      };
      break;

    case OPTUNCHK_TIME:
      {

        //System.out.println("process, optunchk_time");

        ev = new SimpleEvent (OPTUNCHK_TIME);
        EDSimulator.add (30000, ev, node, pid);
        int lucky = CommonState.r.nextInt (nNodes);
        while (cache[byPeer[lucky].peer].status == 1)      // until the lucky peer is
already unchoked
          lucky = CommonState.r.nextInt (nNodes);
        if (!alive (cache[byPeer[lucky].peer].node))
          return;
        cache[byPeer[lucky].peer].status = 1;
        Object msg = new SimpleMsg (UNCHOKE, node);
        latency =
          ((Transport) node.getProtocol (tid)).getLatency (node,
                                                           cache[byPeer
                                                                 [lucky].
                                                                 peer].
                                                           node);
        EDSimulator.add (latency, msg, cache[byPeer[lucky].peer].node, pid);
        cache[byPeer[lucky].peer].justSent ();
      };
      break;

    case ANTISNUB_TIME:
      {
        if (this.peerStatus == 1)   // I'm a seeder, I don't update the event
          return;
        //System.out.println("process, antisnub_time");
        for (int i = 0; i < nNodes; i++)
          {
          if (byPeer[i].valueDOWN > 0
                && (byPeer[i].valueDOWN - byPeer[i].head60) == 0)
              {              // No blocks downloaded in 1 min
                cache[byPeer[i].peer].status = 2;   // I'm snubbed by it
              }
            byPeer[i].head60 = byPeer[i].valueDOWN;
          }
        ev = new SimpleEvent (ANTISNUB_TIME);
        EDSimulator.add (60000, ev, node, pid);
        long time = CommonState.getTime ();
      };
      break;

    case CHECKALIVE_TIME:
      {

        //System.out.println("process, checkalive_time");

        long now = CommonState.getTime ();
        for (int i = 0; i < swarmSize; i++)
          {
            /*If at least 2 minutes (plus 1 sec of tolerance) that
               I don't send anything to it. */
            if (alive (cache[i].node)
                && (cache[i].lastSent < (now - 121000)))
              {
                Object msg = new IntMsg (KEEP_ALIVE, node, 0);
                latency =
                  ((Transport) node.getProtocol (tid)).getLatency (node,
                                                                    cache[i].
                                                                    node);
                EDSimulator.add (latency, msg, cache[i].node, pid);
                cache[i].justSent ();
              }
            /*If at least 2 minutes (plus 1 sec of tolerance) that I don't
                receive anything from it though I sent a keepalive 2 minutes ago */
            else
```

```
                {
                  if (cache[i].lastSeen < (now - 121000)
                      && cache[i].node != null
                      && cache[i].lastSent < (now - 121000))
                    {
                      System.out.
                        println ("process, checkalive_time, rm neigh " +
                                 cache[i].node.getID ());
                      if (cache[i].node.getIndex () != -1)
                        {
                          System.out.
                            println
                            ("This should never happen: I remove a node that did not
effectively die");
                        }
                      removeNeighbor (cache[i].node);
                      processNeighborListSize (node, pid);
                    }
                }
            }
          ev = new SimpleEvent (CHECKALIVE_TIME);
          EDSimulator.add (120000, ev, node, pid);
        };
        break;

      case TRACKERALIVE_TIME:
        {
          //System.out.println("process, trackeralive_time");
          if (alive (tracker))
            {
              ev = new SimpleEvent (TRACKERALIVE_TIME);
              EDSimulator.add (1800000, ev, node, pid);
            }
          else
            tracker = null;

        };
        break;

      case DOWNLOAD_COMPLETED:
        {
          nPiecesUp--;
        };
        break;

      }
  }

        /**
         *      Given a piece index and a block index it encodes them in an unique integer
value.
         *      @param piece the index of the piece to encode.
         *      @param block the index of the block to encode.
         *      @return the encoding of the piece and the block indexes.
         */
  private int encode (int piece, int block)
  {
    return (piece * 100) + block;

  }
        /**
         *      Returns either the piece or the block that contained in the <tt>value</tt>
depending
         *      on <tt>part</tt>: 0 means the piece value, 1 the block value.
         *      @param value the ID of the block to decode.
         *      @param part the information to extract from <tt>value</tt>. 0 means the
piece index, 1 the block index.
         *      @return the piece or the block index depending about the value of
<tt>part</tt>
         */
  private int decode (int value, int part)
  {
    if (value == -1)          // Not a true value to decode
      return -1;
    if (part == 0)            // I'm interested in the piece
      return value / 100;
    else                      // I'm interested in the block
      return value % 100;
  }

        /**
         *     Used by {@link NodeInitializer#choosePieces(int, BitTorrent)
NodeInitializer} to set
         *     the number of piece completed from the beginning according to
         *     the distribution in the configuration file.
         *     @param number the number of piece completed
         */
  public void setCompleted (int number)
  {
    this.nPieceCompleted = number;
  }

        /**
         *     Sets the status (the set of blocks) of the file for the current node.
         *  Note that a piece is considered <i>completed</i> if the number
         *  of downloaded block is 16.
         *  @param index The index of the piece
         *  @param value Number of blocks downloaded for the piece index.
         */
  public void setStatus (int index, int value)
  {
    status[index] = value;
  }

        /**
         *     Sets the status of the local node.
         *     @param status The status of the node: 1 means seeder, 0 leecher
         */
  public void setPeerStatus (int status)
  {
    this.peerStatus = status;
  }

        /**
         *     Gets the status of the local node.
         *     @return The status of the local node: 1 means seeder, 0 leecher
         */
  public int getPeerStatus ()
  {
    return peerStatus;
  }

        /**
         *  Gets the number of blocks for a given piece owned by the local node.
         *     @param index The index of the piece
         *     @return Number of blocks downloaded for the piece index
         */
  public int getStatus (int index)
  {
    return status[index];
  }

        /**
         *     Sets the maximum bandwidth for the local node.
         *     @param value The value of bandwidth in Kbps
         */
  public void setBandwidth (int value)
  {
    maxBandwidth = value;
  }

        /**
         *     Checks if a node is still alive in the simulated network.
         *     @param node The node to check
         *     @return true if the node <tt>node</tt> is up, false otherwise
         *     @see peersim.core.GeneralNode#isUp
         */
  public boolean alive (Node node)
  {
    if (node == null)
      return false;
    else
```

```
      return node.isUp ();
  }

        /**
         *     Adds a neighbor to the cache of the local node.
         *  The new neighbor is put in the first null position.
         *     @param neighbor The neighbor node to add
         *  @return <tt>false</tt> if the neighbor is already present in the cache (this
can happen when the peer requests a
         *     new peer set to the tracker and there is still this neighbor within) or no
place is available.
         *     Otherwise, returns true if the node is correctly added to the cache.
         */
  public boolean addNeighbor (Node neighbor)
  {
    if (search (neighbor.getID ()) != null)
      {                       // if already exists
       //      System.err.println("Node "+neighbor.getID() + " not added, already
exist.");
       return false;
      }
    if (this.tracker == null)
      {                       // I'm in the tracker's BitTorrent protocol
       for (int i = 0; i < nMaxNodes + maxGrowth; i++)
         {
           if (cache[i].node == null)
             {
               cache[i].node = neighbor;
               cache[i].status = 0;  //choked
               cache[i].interested = -1;     //not interested
               this.nNodes++;

               //System.err.println("i: " + i +" nMaxNodes: " + nMaxNodes);
               return true;
             }
         }
      }
    else
      {
       if ((nNodes + nBitfieldSent) < swarmSize)
         {
           //System.out.println("I'm the node " + this.thisNodeID + ", trying to add node
"+neighbor.getID());
           for (int i = 0; i < swarmSize; i++)
             {
               if (cache[i].node == null)
                 {
                   cache[i].node = neighbor;
                   cache[i].status = 0;      //choked
                   cache[i].interested = -1; // not interested
                   byPeer[nNodes].peer = i;
                   byPeer[nNodes].ID = neighbor.getID ();
                   sortByPeer ();
                   this.nNodes++;
                   //System.out.println(neighbor.getID()+" added!");
                   return true;
                 }
             }
           System.out.println ("Node not added, no places available");
         }
      }
    return false;
  }

        /**
         *     Removes a neighbor from the cache of the local node.
         *     @param neighbor The node to remove
         *     @return true if the node is correctly removed, false otherwise.
      */
  public boolean removeNeighbor (Node neighbor)
  {

    if (neighbor == null)
      return true;

    // this is the tracker's bittorrent protocol
    if (this.tracker == null)
      {
        for (int i = 0; i < (nMaxNodes + maxGrowth); i++)
          {

           // check the feasibility of the removal
           if ((cache[i] != null) && (cache[i].node != null) &&
               (cache[i].node.getID () == neighbor.getID ()))
             {
               cache[i].node = null;
               this.nNodes--;
               return true;
             }
          }
        return false;
      }
    // this is the bittorrent protocol of a peer
    else
      {

        Element e = search (neighbor.getID ());

        if (e != null)
          {
           for (int i = 0; i < nPieces; i++)
             {
               rarestPieceSet[i] -= swarm[e.peer][i];
               swarm[e.peer][i] = 0;
             }

           cache[e.peer].node = null;
           cache[e.peer].status = 0;
           cache[e.peer].interested = -1;
           unchokedBy[e.peer] = false;
           this.nNodes--;
           e.peer = -1;
           e.ID = Integer.MAX_VALUE;
           e.valueUP = 0;
           e.valueDOWN = 0;
           e.head20 = 0;
           e.head60 = 0;
           sortByPeer ();

           return true;
          }
      }
    return false;
  }

        /**
      *  Adds a request to the pendingRequest queue.
         *     @param block The requested block
         *     @return true if the request has been successfully added to the queue,
false otherwise
         */
  private boolean addRequest (int block)
  {
    int i = 4;
    while (i >= 0 && pendingRequest[i] != -1)
      {
        i--;
      }
    if (i >= 0)
      {
        pendingRequest[i] = block;
        return true;
      }
      else
      {                       // It should never happen
        //System.err.println("pendingRequest queue full");
        return false;
      }
  }

        /**
```

```
         *      Removes the block with the given <tt>id</tt> from the {@link
#pendingRequest} queue
         *  and sorts the queue leaving the empty cell at the left.
         *      @param id the id of the requested block
         */
  private void removeRequest (int id)
  {
    int i = 4;
    for (; i >= 0; i--)
      {
        if (pendingRequest[i] == id)
          break;
      }
    for (; i >= 0; i--)
      {
        if (i == 0)
          pendingRequest[i] = -1;
        else
          pendingRequest[i] = pendingRequest[i - 1];
      }
  }

        /**
         *      Requests new block until the {@link #pendingRequest} is full to the sender
of the just received piece.
         *      It calls {@link #getNewBlock(Node, int)} to implement the <i>strict
priority</i> strategy.
         *      @param node the local node
         *      @param pid the BitTorrent protocol id
         *      @param sender the sender of the just received piece.
         */
  private void requestNextBlocks (Node node, int pid, int sender)
  {
    int block = getNewBlock (node, pid);
    while (block != -2)
      {
        if (unchokedBy[sender] == true && alive (cache[sender].node)
            && addRequest (block))
          {
            Object ev = new IntMsg (REQUEST, node, block);
            long latency =
              ((Transport) node.getProtocol (tid)).getLatency (node,
                                                               cache[sender].
                                                               node);
            EDSimulator.add (latency, ev, cache[sender].node, pid);
            cache[sender].justSent ();
          }
        else
          {                    // I cannot send request
            if (!alive (cache[sender].node) && cache[sender].node != null)
              {
                System.out.println ("piece2 rm neigh " +
                                    cache[sender].node.getID ());
                removeNeighbor (cache[sender].node);
                processNeighborListSize (node, pid);
              }
            return;
          }
        block = getNewBlock (node, pid);
      }
  }

        /**
         *      It returns the id of the next block to request. Sends <tt>INTERESTED</tt>
if the new
         *      block belongs to a new piece.
         *      It uses {@link #getBlock()} to get the next block of a piece and calls
{@link #getPiece()}
         *      when all the blocks for the {@link #currentPiece} have been requested.
         *      @param node the local node
         *      @param pid the BitTorrent protocol id
         *      @return -2 if no more places available in the <tt>pendingRequest</tt>
queue;<br/>
         *                      the value of the next block to request otherwise</p>
         */
  private int getNewBlock (Node node, int pid)
  {
    int block = getBlock ();
    if (block < 0)
      {                       // No more blocks to request for the current piece

        if (block == -2)      // Pending request queue full
          return -2;

        int newPiece = getPiece ();
        if (newPiece == -1)
          {                   // no more pieces to request
            return -2;
          }

        lastInterested = newPiece;
        Object ev = new IntMsg (INTERESTED, node, lastInterested);

        for (int j = 0; j < swarmSize; j++)
          {                   // send the interested message to those
            // nodes which have that piece
            if (alive (cache[j].node) && swarm[j][newPiece] == 1)
              {
                long latency =
                  ((Transport) node.getProtocol (tid)).getLatency (node,
                                                                   cache[j].
                                                                   node);
                EDSimulator.add (latency, ev, cache[j].node, pid);
                cache[j].justSent ();
              }
            if (!alive (cache[j].node))
              {
                //System.out.println("piece1 rm neigh "+ cache[j].node.getID() );

                removeNeighbor (cache[j].node);
                processNeighborListSize (node, pid);
              }
          }
        block = getBlock ();
        return block;
      }
    else
      {
        // block value referred to a real block
        return block;
      }
  }

        /**
         *     Returns the next block to request for the {@link #currentPiece}.
         *     @return an index of a block of the <tt>currentPiece</tt> if there are
still
         *                    available places in the {@link #pendingRequest} queue;<br/>
         *                    -2 if the <tt>pendingRequest</tt> queue is full;<br/>
         *                    -1 if no more blocks to request for the current piece.
         */
  private int getBlock ()
  {
    int i = 4;
    while (i >= 0 && pendingRequest[i] != -1)
      {                       // i is the first empty position from the head
        i--;
      }
    if (i == -1)
      {                       // No places in the pendingRequest available
        //System.out.println("Pendig request queue full!");
        return -2;
      }
    int j;
    //The queue is not empty & last requested block belongs to lastInterested piece
    if (i != 4 && decode (pendingRequest[i + 1], 0) == lastInterested)
      j = decode (pendingRequest[i + 1], 1) + 1;    // the block following the last
requested
    else                      // I don't know which is the next block, so I search it.
      j = 0;
    /*      I search another block until the current has been already received.
     *      If in pieceStatus at position j there is a block that belongs to
     *      lastInterested piece, means that the block j has been already
     *      received, otherwise I can request it.
     */
    while (j < 16 && decode (pieceStatus[j], 0) == lastInterested)
      {
        j++;
      }
    if (j == 16)               // No more block to request for lastInterested piece
      return -1;
    return encode (lastInterested, j);
  }

        /**
         *     Returns the next correct piece to download. It chooses the piece by using
the
         *     <i>random first</i> and <i>rarest first</i> policy. For the beginning 4
pieces
         *     of a file the first one is used then the pieces are chosen using <i>rarest
first</i>.
         *     @see "Documentation about the BitTorrent module"
         *     @return the next piece to download. If the whole file has been requested
         *     -1 is returned.
         */
  private int getPiece ()
  {
    int piece = -1;
    if (nPieceCompleted < 4)
      {                        //Uses random first piece
        piece = CommonState.r.nextInt (nPieces);
        while (status[piece] == 16 || piece == currentPiece) // until the piece is owned
          piece = CommonState.r.nextInt (nPieces);
        return piece;
      }
    else
      {                        //Uses rarest piece first
        int j = 0;
        for (; j < nPieces; j++)
          {                    // I find the first not owned piece
            if (status[j] == 0)
              {
                piece = j;
                if (piece != lastInterested)  // teoretically it works because
                  // there should be only one interested
                  // piece not yet downloaded
                  break;
              }
          }
        if (piece == -1)
          {                    // Never entered in the previous 'if' statement; for all
            // pieces an has been sent
            return -1;
          }

        int rarestPieces[] = new int[nPieces - j];   // the pieces with the less number of
occurrences\
        rarestPieces[0] = j;
        int nValues = 1;       // number of pieces less distributed in the network
        for (int i = j + 1; i < nPieces; i++)
          {                    // Finds the rarest piece not owned
            if (rarestPieceSet[i] < rarestPieceSet[rarestPieces[0]]
                && status[i] == 0)
              {                // if strictly less than the current one
                rarestPieces[0] = i;
                nValues = 1;
              }
            if (rarestPieceSet[i] == rarestPieceSet[rarestPieces[0]]
                && status[i] == 0)
              {                // if equal
                rarestPieces[nValues] = i;
                nValues++;
              }
          }

        piece = CommonState.r.nextInt (nValues);     // one of the less owned pieces
        return rarestPieces[piece];
      }
  }

        /**
         *     Returns the file's size as number of pieces of 256KB.
         *     @return number of pieces that compose the file.
         */
  public int getNPieces ()
  {
    return nPieces;
  }
        /**
         *     Clone method of the class. Returns a deep copy of the BitTorrent class.
Used
         *     by the simulation to initialize the {@link peersim.core.Network}
         *     @return the deep copy of the BitTorrent class.
         */
  public Object clone ()
  {
    Object prot = null;
    try
    {
      prot = (BitTorrent) super.clone ();
    }
    catch (CloneNotSupportedException e)
    {
    };

    ((BitTorrent) prot).cache = new Neighbor[swarmSize];
    for (int i = 0; i < swarmSize; i++)
      {
        ((BitTorrent) prot).cache[i] = new Neighbor ();
      }

    ((BitTorrent) prot).byPeer = new Element[swarmSize];
    for (int i = 0; i < swarmSize; i++)
      {
        ((BitTorrent) prot).byPeer[i] = new Element ();
      }

    ((BitTorrent) prot).unchokedBy = new boolean[swarmSize];

    ((BitTorrent) prot).byBandwidth = new Element[swarmSize];
    ((BitTorrent) prot).status = new int[nPieces];
    ((BitTorrent) prot).pieceStatus = new int[16];
    for (int i = 0; i < 16; i++)
      ((BitTorrent) prot).pieceStatus[i] = -1;
    ((BitTorrent) prot).pendingRequest = new int[5];
    for (int i = 0; i < 5; i++)
      ((BitTorrent) prot).pendingRequest[i] = -1;
    ((BitTorrent) prot).rarestPieceSet = new int[nPieces];
    for (int i = 0; i < nPieces; i++)
      ((BitTorrent) prot).rarestPieceSet[i] = 0;
    ((BitTorrent) prot).swarm = new int[swarmSize][nPieces];
    ((BitTorrent) prot).requestToServe = new Queue (20);
    ((BitTorrent) prot).incomingPieces = new Queue (100);
    return prot;
  }

        /**
         *     Sorts {@link #byPeer} array by peer's ID. It implements the
<i>InsertionSort</i>
         *     algorithm.
         */
  public void sortByPeer ()
  {
    int i;

    for (int j = 1; j < swarmSize; j++)      // out is dividing line
      {
        Element key = new Element ();
        byPeer[j].copyTo (key);       // remove marked item
        i = j - 1;             // start shifts at out
        while (i >= 0 && (byPeer[i].ID > key.ID))    // until one is smaller,
          {
            byPeer[i].copyTo (byPeer[i + 1]); // shift item right,
            i--;               // go left one position
        }
      key.copyTo (byPeer[i + 1]);   // insert marked item
    }

  }

        /**
         *     Sorts the array {@link #byBandwidth} using <i>QuickSort</i> algorithm.
         *     <tt>null</tt> elements and seeders are moved to the end of the array.
         */
  public void sortByBandwidth ()
  {
    quicksort (0, swarmSize - 1);
  }

        /**
         *     Used by {@link #sortByBandwidth()}. It's the implementation of the
         *     <i>QuickSort</i> algorithm.
         *     @param left the leftmost index of the array to sort.
         *     @param right the rightmost index of the array to sort.
         */
  private void quicksort (int left, int right)
  {
    if (right <= left)
      return;
    int i = partition (left, right);
    quicksort (left, i - 1);
    quicksort (i + 1, right);
  }

        /**
         *     Used by {@link #quicksort(int, int)}, partitions the subarray to sort
returning
         *     the splitting point as stated by the <i>QuickSort</i> algorithm.
         *     @see "The <i>QuickSort</i> algorithm".
         */
  private int partition (int left, int right)
  {
    int i = left - 1;
    int j = right;
    while (true)
      {
        while (greater (byBandwidth[++i], byBandwidth[right]))      // find item on left
to swap
          ;                    // a[right] acts as sentinel
        while (greater (byBandwidth[right], byBandwidth[--j]))
          {                    // find item on right to swap
            if (j == left)
              break;           // don't go out-of-bound
          }
        if (i >= j)
          break;               // check if pointers cross
        swap (i, j);           // swap two elements into place
      }
    swap (i, right);           // swap with partition element
    return i;
  }

        /**
         *     Aswers to the question "is x > y?". Compares the {@link Element}s given as
         *     parameters. <tt>Element x</tt> is greater than <tt>y</tt> if isn't
<tt>null</tt>
         *     and in the last 20 seconds the local node has downloaded ("uploaded" if
the local node is a
         *     seeder) more blocks than from <tt>y</tt>.
         *     @param x the first <tt>Element</tt> to compare.
         *     @param y the second <tt>Element</tt> to compare
         *     @return <tt>true</tt> if x > y;<br/>
         *                    <tt>false</tt> otherwise.
         */
  private boolean greater (Element x, Element y)
  {
    /*
     * Null elements and seeders are shifted at the end
     * of the array
     */
    if (x == null)
      return false;
    if (y == null)
      return true;
    if (x.isSeeder)
      return false;
    if (y.isSeeder)
      return true;

    // if the local node is a leecher
    if (peerStatus == 0)
      {
       if ((x.valueDOWN - x.head20) > (y.valueDOWN - y.head20))
         return true;
       else
         return false;
      }

    // if peerStatus==1 (the local node is a seeder)
    else
      {
       if ((x.valueUP - x.head20) > (y.valueUP - y.head20))
         return true;
       else
         return false;
      }
  }

       /**
        *     Swaps {@link Element} <tt>i</tt> with <tt>j</tt> in the {@link
#byBandwidth}.<br/>
        *     Used by {@link #partition(int, int)}
        *     @param i index of the first element to swap
        *     @param j index of the second element to swap
        */
  private void swap (int i, int j)
  {
    Element swap = byBandwidth[i];
    byBandwidth[i] = byBandwidth[j];
    byBandwidth[j] = swap;
  }

       /**    Searches the node with the given ID. It does a dychotomic
        *     search.
        *     @param ID the ID of the node to search.
        *     @return the {@link Element} in {@link #byPeer} which represents the node
with the
        *     given ID.
        */
  public Element search (long ID)
  {
    int low = 0;
    int high = swarmSize - 1;
    int p = low + ((high - low) / 2); //Initial probe position
    while (low <= high)
      {
       if (byPeer[p] == null || byPeer[p].ID > ID)
         high = p - 1;
       else
         {
           if (byPeer[p].ID < ID)    //Wasteful second comparison forced by syntax
limitations.
             low = p + 1;
           else
             return byPeer[p];
         }
       p = low + ((high - low) / 2); //Next probe position.
      }
    return null;
  }
}

/**
 *     This class is used to store the main information about a neighbors regarding
 *     the calculation of the Downloading/Uploading rates. Is the class of items in
 *      {@link example.bittorrent.BitTorrent#byPeer} and {@link
example.bittorrent.BitTorrent#byBandwidth}.
 */
class Element
{
        /**
         *      ID of the represented node.
         */
  public long ID = Integer.MAX_VALUE;
        /**
         *      Index position of the node in the {@link
example.bittorrent.BitTorrent#cache} array.
         */
  public int peer = -1;
        /**
         *      Number of blocks uploaded to anyone since the beginning.
         */
  public int valueUP = 0;
        /**
         *      Number of blocks downloaded from anyone since the beginning.
         */
  public int valueDOWN = 0;
        /**
         *      Value of either {@link #valueUP} or {@link #valueDOWN} (depending by
         *      {@link example.bittorrent.BitTorrent#peerStatus}) 20 seconds before.
         */
  public int head20 = 0;
        /**
         *      Value of either {@link #valueUP} or {@link #valueDOWN} (depending by
         *      {@link example.bittorrent.BitTorrent#peerStatus}) 60 seconds before.
         */
  public int head60 = 0;
        /**
         *      <tt>true</tt> if the node is a seeder, <tt>false</tt> otherwise.
         */
  public boolean isSeeder = false;
        /**
         *      Makes a deep copy of the Element to <tt>destination</tt>
         *      @param destination Element instance where to make the copy
         */
  public void copyTo (Element destination)
  {
    destination.ID = this.ID;
    destination.peer = this.peer;
    destination.valueUP = this.valueUP;
    destination.valueDOWN = this.valueDOWN;
    destination.head20 = this.head20;
    destination.head60 = this.head60;
  }
}

/**
 *      This class stores information about the neighbors regarding their status. It is
 *      the type of the items in the {@link example.bittorrent.BitTorrent#cache}.
 */
class Neighbor
{
        /**
         *      Reference to the node in the {@link peersim.core.Network}.
         */
  public Node node = null;
        /**
         *      -1 means not interested<br/>
         *      Other values means the last piece number for which the node is interested.
         */
  public int interested;
        /**
         *      0 means CHOKED<br/>
         *      1 means UNCHOKED<br/>
         *      2 means SNUBBED_BY. If this value is set and the node is to be unchoked,
         *      value 2 has the priority.
         */
  public int status;
        /**
         *      Last time a message from the node represented has been received.
         */
  public long lastSeen = 0;
        /**
         *     Last time a message to the node represented has been sent.
         */
  public long lastSent = 0;

        /**
         * Sets the last time the neighbor was seen.
         */
  public void isAlive ()
  {
    long now = CommonState.getTime ();
      this.lastSeen = now;
  }

  /*
   * Sets the last time the local peer sent something to the neighbor.
   */
  public void justSent ()
  {
    long now = CommonState.getTime ();
    this.lastSent = now;
  }

}

/**
 *     Class type of the queues's items in {@link
example.bittorrent.BitTorrent#incomingPieces}
 *     and {@link example.bittorrent.BitTorrent#requestToServe}.
 */
class Queue
{
  int maxSize;
  int head = 0;
  int tail = 0;
  int dim = 0;
  Request queue[];

        /**
         *     Public constructor. Creates a queue of size <tt>size</tt>.
         */
  public Queue (int size)
  {
    maxSize = size;
    queue = new Request[size];
    for (int i = 0; i < size; i++)
      queue[i] = new Request ();
  }

        /**
         *     Enqueues the request of the block <tt>id</tt> and its <tt>sender</tt>
         *     @param id the id of the block in the request
         *     @param sender a reference to the sender of the request
         *     @return <tt>true</tt> if the request has been correctly added,
<tt>false</tt>
         *     otherwise.
         */
  public boolean enqueue (int id, Node sender)
  {
    if (dim < maxSize)
      {
        queue[tail % maxSize].id = id;
        queue[tail % maxSize].sender = sender;
        tail++;
        dim++;
        return true;
      }
    else
      return false;
  }

        /**
         *     Returns the {@link Request} in the head of the queue.
         *     @return the element in the head.<br/>
         *                     <tt>null</tt> if the queue is empty.
         */
  public Request dequeue ()
  {
    Request value;
    if (dim > 0)
      {
        value = queue[head % maxSize];
        head++;
        dim--;
        return value;
      }
    else
      return null;            //empty queue
  }

        /**
         *     Returns the status of the queue.
         *     @return <tt>true</tt> if the queue is empty, <tt>false</tt>
         *     otherwise.
         */
  public boolean empty ()
  {
    return (dim == 0);
  }

        /**
         *     Returns <tt>true</tt> if block given as parameter is in.
         *     @param  value the id of the block to search.
         *     @return <tt>true</tt> if the block <tt>value</tt> is in the queue,
<tt>false</tt>
         *     otherwise.
         */
  public boolean contains (int value)
  {
    if (empty ())
      return false;
    for (int i = head; i < head + dim; i++)
      {
        if (queue[i % maxSize].id == value)
          return true;
      }
    return false;
  }

        /**
         *     Removes a request from the queue.
         *     @param sender the sender of the request.
         *     @param value the id of the block requested.
         *     @return <tt>true</tt> if the request has been correctly removed,
<tt>false</tt>
         *     otherwise.
         */
  public boolean remove (Node sender, int value)
  {
    if (empty ())
      return false;
    for (int i = head; i < head + dim; i++)
      {
        if (queue[i % maxSize].id == value
            && queue[i % maxSize].sender == sender)
          {
            for (int j = i; j > head; j--)
              {              // Shifts the elements for the removal
                queue[j % maxSize] = queue[(j - 1) % maxSize];
              }
            head++;
            dim--;
            return true;
          }
      }
    return false;
  }
}

/**
 *     This class represent an enqueued request of a block.
 */
class Request
{
        /**
         *     The id of the block.
         */
  public int id;
        /**
         *     The sender of the request.
         */
  public Node sender;
}
```

This program (PeerSim) was used to initialize the node. The link capacity parameter values were set in the code

.

```
/*
 * Copyright (c) 2007-2008 Fabrizio Frioli, Michele Pedrolli
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU Lesser General Public License version 2 as
 * published by the Free Software Foundation.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

package peersim.bittorrent;

import peersim.core.*;
import peersim.config.Configuration;

/**
 *     This class provides a way to initialize a single node of the network.
 *     The initialization is performed by choosing the bandwidth of the node
 *     and choosing how much the shared file has been downloaded.
 */
public class NodeInitializer
{

        /**
         *     The protocol to operate on.
         *     @config
         */
  private static final String PAR_PROT = "protocol";

        /**
         *     The percentage of nodes with no downloaded pieces.
         *     @config
         *     @see "The documentation for an example on how to properly set this
parameter."
         */
  private static final String PAR_NEWER_DISTR = "newer_distr";

        /**
         *     The percentage of seeders in the network.
         *     @config
         */
  private static final String PAR_SEEDER_DISTR = "seeder_distr";

        /**
         *     The percentage of nodes with no downloaded pieces,
         *     as defined in {@see #PAR_NEWER_DISTR}.
         */
  private int newerDistr;

        /**
         *     The percentage of seeder nodes,
         *     as defined in {@see #PAR_SEEDER_DISTR}.
         */
  private int seederDistr;

        /**
         *     The BitTorrent protocol ID.
         */
  private final int pid;

        /**
         *     The basic constructor of the class, which reads the parameters
         *      from the configuration file.
         *      @param prefix the configuration prefix for this class
         */
  public NodeInitializer (String prefix)
  {
    pid = Configuration.getPid (prefix + "." + PAR_PROT);
    newerDistr = Configuration.getInt (prefix + "." + PAR_NEWER_DISTR);
    seederDistr = Configuration.getInt (prefix + "." + PAR_SEEDER_DISTR);
  }

        /**
         *      Initializes the node <tt>n</tt> associating it
         *      with the BitTorrent protocol and setting the reference to the tracker,
         *      the status of the file and the bandwidth.
         *      @param n The node to initialize
         */
  public void initialize (Node n)
  {
    Node tracker = Network.get (0);
    BitTorrent p;
    p = (BitTorrent) n.getProtocol (pid);
    p.setTracker (tracker);
    p.setThisNodeID (n.getID ());
    setFileStatus (p);
    setBandwidth (p);
  }

        /**
         *      Sets the status of the shared file according to the
         *      probability value given by {@link #getProbability()}.
         *      @param p The BitTorrent protocol
         */
  private void setFileStatus (BitTorrent p)
  {
    int percentage = getProbability ();
    choosePieces (percentage, p);
  }

        /**
         *      Set the maximum bandwidth for the node.
         *      @param p The BitTorrent protocol
         */
// For 100 kb/s link speed
  private void setBandwidth (BitTorrent p)
  {
    int value = CommonState.r.nextInt (4);
    switch (value)
      {
      case 0:
        p.setBandwidth (100);
        break;
      case 1:
        p.setBandwidth (100);
        break;
      case 2:
        p.setBandwidth (100);
        break;
      case 3:
        p.setBandwidth (100);
        break;
      }
  }

// For 200 kb/s link speed
  private void setBandwidth (BitTorrent p)
  {
    int value = CommonState.r.nextInt (4);
    switch (value)
      {
      case 0:
        p.setBandwidth (200);
        break;
      case 1:
        p.setBandwidth (200);
        break;
      case 2:
        p.setBandwidth (200);
        break;
      case 3:
        p.setBandwidth (200);
        break;
      }
  }

// For 50 kb/s link speed
  private void setBandwidth (BitTorrent p)
  {
    int value = CommonState.r.nextInt (4);
    switch (value)
      {
      case 0:
        p.setBandwidth (50);
        break;
      case 1:
        p.setBandwidth (50);
        break;
      case 2:
        p.setBandwidth (50);
        break;
      case 3:
        p.setBandwidth (50);
        break;
      }
  }

        /**
         *     Sets the completed pieces for the given protocol <tt>p</tt>.
         *     @parm percentage The percentage of the downloaded pieces, according to
{@link #getProbability()}
         *     @param p the BitTorrent protocol
         */
  private void choosePieces (int percentage, BitTorrent p)
  {
    double temp = ((double) p.nPieces / 100.0) * percentage;       // We use a double to
avoid the loss of precision
    // during the division operation
    int completed = (int) temp;      //integer number of piece to set as completed
    //0 if the peer is a newer
    p.setCompleted (completed);
    if (percentage == 100)
      p.setPeerStatus (1);
    int tmp;
    while (completed != 0)
      {
        tmp = CommonState.r.nextInt (p.nPieces);
        if (p.getStatus (tmp) != 16)
          {
            p.setStatus (tmp, 16);
            completed--;
          }
      }
  }

        /**
         *     Gets a probability according with the parameter <tt>newer_distr</tt>
         *     defined in the configuration file.
         *     @return the probabilty value, where 0 means that the peer is new and no
pieces has been downloaded,
         *                   100 means that the peer is a seeder; other values defines a
random probability.
         *     @see #PAR_NEWER_DISTR
         */
  private int getProbability ()
  {
    int value = CommonState.r.nextInt (100);
    if ((value + 1) <= seederDistr)
      return 100;
    value = CommonState.r.nextInt (100);
    if ((value + 1) <= newerDistr)
      {
        return 0;              // A newer peer, with probability newer_distr
      }
    else
      {
       value = CommonState.r.nextInt (9);
       return (value + 1) * 10;
      }
  }
}
```

This script (PeerSim) was used to change values of the different parameters for the simulations.

Command to run the script is:

```
make all > name of output file
```

```
#Config file for BitTorrent extension

random.seed 1234567890
simulation.endtime 10 ^ 9
simulation.logtime 10 ^ 3

simulation.experiments 1000

network.size 500
network.node peersim.core.GeneralNode

protocol.urt UniformRandomTransport
protocol.urt.mindelay 5
protocol.urt.maxdelay 20

protocol.bittorrent peersim.bittorrent.BitTorrent
protocol.bittorrent.file_size 100
protocol.bittorrent.max_swarm_size 80
protocol.bittorrent.peerset_size 50
protocol.bittorrent.duplicated_requests 1
protocol.bittorrent.transport urt
protocol.bittorrent.max_growth 20

init.net peersim.bittorrent.NetworkInitializer
init.net.protocol bittorrent
init.net.transport urt
init.net.newer_distr 20
init.net.seeder_distr 1

control.observer peersim.bittorrent.BTObserver
control.observer.protocol bittorrent
control.observer.step 10000

control.dynamics peersim.bittorrent.NetworkDynamics
control.dynamics.protocol bittorrent
control.dynamics.newer_distr 20
control.dynamics.minsize 20
control.dynamics.tracker_can_die 0
control.dynamics.step 10000
control.dynamics.transport urt
control.dynamics.add 5
control.dynamics.remove 5
```

CURRICULUM VITAE

Khondkar R. Islam is a native of Bangladesh. He received the Bachelor of Science in Economics from George Mason University and a Master of Science in Computer Science (CIS) from American University in Washington, DC. He was awarded the George Mason Doctoral Fellowship award in 2004, 2005 and 2006. He has over 15 years of experience working in Information Technology and Telecommunications, National Aeronautics and Space Administration and Department of Homeland Security government agencies, as well as academia at George Mason University, University of Maryland at College Park and American University. Dr. Islam is also a subject matter expert at Prometric. He has reviewed several technical book chapters for Prentice Hall, McGraw-Hill, John Wiley & Sons, and Addison-Wesley, and has published several articles.