# IMPROVING BITTORRENT CHOKING ALGORITHM TO MITIGATE FREE RIDING

BY

**MURTADA IBRAHIM AL-HABIB**

A Thesis Presented to the

DEANSHIP OF GRADUATE STUDIES

**KING FAHD UNIVERSITY OF PETROLEUM & MINERALS**

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

# MASTER OF SCIENCE

In

**COMPUTER SCIENCE**

**May 2014**

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN- 31261, SAUDI ARABIA

**DEANSHIP OF GRADUATE STUDIES**

This thesis, written by **Murtada Ibrahim Al-Habib** under the direction of his thesis advisor and approved by his thesis committee, has been presented and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN COMPUTER SCIENCE.**

Dr. FARAG AZZEDIN
(Advisor)

Dr. ADEL FADHL AHMED
Department Chairman

Dr. SAJJAD MAHMOOD
(Member)

Dr. SALAM A. ZUMMO
Dean of Graduate Studies

Dr. SAMI ZHIOUA
(Member)

8/5/14
Date

This thesis is dedicated to my parents, wife, family and friends

# ACKNOWLEDGMENTS

First of all, thanks to God for all the blessings, including the support and influence to successfully complete this thesis. Thanks to my advisor, Dr. Farag Azzedin for his continues support and assistance before and during this research. This research wouldn't be as successful without Dr. Farag's commitment and dedication. Special thanks to the research committee, Dr. Sami Zhioua and Dr. Sajjad Mahmood for their contribution in assisting this research throughout all of its phases.

I deeply appreciate the continuous encouragement of my supervisors, administrators and the manager of Exploration Applications Services Department in Saudi ARAMCO. Thanks to Mr. Adel Naji, Mr. Abdullah Eidi and Mr. Ahmed Ramadan for providing all the needed support throughout this research period. Special thanks go to all individuals in Saudi ARAMCO who supported and encouraged me to complete this degree, especially Ali Salem who provided an outstanding support during the early phases of my enrollment to this degree.

No words can express my deep appreciation to my parents, wife, my two brothers and sister for all the patience, prayers and good wishes. Without them, it won't be possible to survive this tough period of working at Saudi ARAMCO, in addition to the course work and being a part time student doing this research after working hours.

Thanks to everybody who provided any sort of support during the entire period of my Masters of Science in Computer Science class work and research.

# TABLE OF CONTENTS

# LIST OF FIGURES

# ABSTRACT

Full Name : Murtada Ibrahim Al-Habib

Thesis Title : IMPROVING BITTORRENT CHOKING ALGORITHM TO MITIGATE FREE RIDING

Major Field : Information and Computer Science

Date of Degree : May, 2014

Peer-to-Peer (P2P) systems are known for their flexibility and scalability. In addition, they are famous for their ability to provide the required infrastructure to share resources in a very efficient manner. In a P2P network, peers are expected to share resources in return of using the network and other peers' resources. Recently, BitTorrent became one of the most popular tools for file sharing over the Internet. BitTorrent is designed for distribution assuming full collaboration between peers.

Many studies showed different achievable free riding attacks to target BitTorrent. An example is BitThief which can increase the number of connections to other peers by sending multiple TRACKER messages to the tracker and get the advantage of connecting to more seeders in the environment. Another example is sending garbage pieces, this is when a free rider sends garbage pieces to trick others that it is providing to the system and therefore be unchoked normally.

We have implemented a new type of free riding through not sending HAVE message and not honestly announcing the file status through the BITFIELD message. This thesis studies the effect of this type of free riding in BitTorrent environment using Peersim simulator. An algorithm to mitigate free riders and slowdown their file download rate is

also developed, through modifying the choking algorithm and the optimistic unchoking algorithm.

The implemented algorithm shows that free riders can be detected and punished for their selfish behavior. The algorithm is based on X chances giving algorithm where all peers are given the opportunity to download blocks without running any history check on them. After X blocks upload, an experience check of leechers and seeders with experience is used to decide wither to unchoke a given peer or not.

The results of the study demonstrated that mitigating free riding in BitTorrent is achievable to a significant extent. It also shows that the percentage of free riders, leechers and seeders in the environment have a great impact on the detection of free riders and the degree to which free riders can be delayed. The larger the percentage of initial seeders in the network who are not downloading put uploading to others, the more free riders can get advantage of the BitTorrent environment. Seeders do not have a mechanism to detect whether other peers have contributed anything to them or not and thus they have no history with them.

# ملخص الرسالة

**الاسم الكامل: / مرتضى ابراهيم الحبيب**

**عنوان الرسالة: تطوير نظام الخنق في بت تورنت للحد من المستخدمين غير المشاركين**

**التخصص: علوم الحاسب الآلي**

**تاريخ الدرجة العلمية: مايو , ٢٠١٤ م**

تشتهر أنظمة الند للند بالفعالية والكفائة العالية لمشاركة المصادر الحاسوبية والملفات. من بين أشهر هذه الأنظمة هو نظام بت تورنت الذي أسس في عام ٢٠٠١ علي يد المبرمج برام كوهين. كانت لغة البرمجه بايثون هي أول اللغات المستخدمة في برمجة أول برتوكول في نظام البت تورنت.

ترجع شهرة بت تورنت إلى قدرته العالية في توفير سرعات عالية لتنزيل ملفات عالية المساحة, وذلك لقدرته على تقسيم عملية التحميل على عدة مستخدمين يتشاركون في طريقة تكاملية على توفير أجزاء الملف المراد تحميله. يعتمد نظام بت تورنت على المشاركة الفعاله, بدون هذه المشاركة, تضعف قوة النظام تنتفي فعاليته ومقدار نزاهته, بالتالي تنتفي أهم خصائصه.

أثبتت الدراسات على تمكن بعض المستخدمين من خداع نظام الخناق في بت تورنت والذي يتحكم بنزاهة توزيع الملفات, معتمدا على كمية الوحدات المحملة من الطرف الآخر في عملية تبادل الوحدات المكونة للملف الأساسي.

تهدف هذه الرسالة البحثية لتطوير نظام الخنق في بت تورنت للتمكن من التقليل من أثر المستخدمين الذين يهدفون لخداع نظام الخنق في بت تورنت وذلك عن طريق إضافة عامل الخبرة في التعاملات السابقه لتحديد ما اذا كان يجب فك الخناق عن مستخدم من عدمه.

أثبتت هذه الدراسه أن بالإمكان تقليل الأثر السلبي لهذا النوع من المستخدمين غير فعالي المشاركة وذلك بتبطئة قدرة التحميل لهم بالمقارنة مع غيرهم من المستخدمين فعالي المشاركة.

# CHAPTER 1

# INTRODUCTION

Peer-to-Peer (P2P) systems are known for their flexibility, scalability, and their ability to provide the required infrastructure to share resources in a very efficient manner. In a P2P network, peers are expected to share resources in return of using the network and other peers' resources [9, 20]. P2P architectures utilization is increasing in recent distributed computation solutions because of the efficiency in performance, low maintenance cost and high scalability [3].

P2P networks can be classified based on different criteria, but the most famous classification criteria are the degree of centralization and network layout. The degree of centralization determines the extent on which the P2P network depends on a central server to control the interaction between the peers. The degree of structure refers to the way the resources are structured, indexed and distributed in the network. Based on the mentioned criteria, the P2P networks can be classified into three major categories: centralized, decentralized but structured (hybrid) and decentralized and unstructured (pure) [9].

In centralized P2P networks, a central directory that is constantly updated is used by all peers to locate and discover resources. An example of centralized P2P network is Napster. In a hybrid network, the peers are controlled in a systematic topology following a certain algorithm. An example of a hybrid P2P network is BitTorrent. In a pure P2P

network, there is no central control and peers have little control over the network. An example of pure P2P network is Kazaa [9].

In general, users of P2P system have the flexibility to choose the peers they would like to connect to and they can set limitations of number of connections and bandwidth capacity of their direct connections. The users' resources such as content and computation power are the heart of each P2P system [3]. Downloading a large file from a server utilizing a regular client server approach will lead to an overhead on the server side, especially when a large number of concurrent connections try to access and download files at the same time [7].

In any P2P network, peers are expected to provide their resources to the rest of the network. Examples of resources are bandwidth, storage and computing power. File sharing is one of the most important applications of P2P networks. This is because of the nature of the P2P networks, which is based on collaboration in a distributed computing environment to solve a problem that is computationally intensive with a minimal chance of failing. In P2P file sharing, all peers are expected to share either a complete file or at least part of it. This is different than the file downloading websites were participation is not required.

In the past few years, BitTorrent became one of the most popular tools for file sharing over the Internet. It can be utilized to share large files without affecting the performance of the origin server [2]. Studies show that around 40% of the Internet traffic in 2004, and 60% in 2006 is due to the BitTorrent file exchange. Also, around 170 million people use BitTorrent services every month [4, 18]. Studies also show that around 78% of P2P

network traffic is due to BitTorrent file exchange traffic. It is also claimed that at any point of time, the users of BitTorrent are more than the users of YouTube and Facebook combined [18]. One key factor of the popularity of BitTorrent is the high achievable download rate by the participating peers.

The first implementation of BitTorrent protocol was designed in April 2001 and implemented in July 2001; it was designed and programmed by Bram Cohen. Currently, BitTorrent is a company that manages and enhances many BitTorrent related products. BitTorrent Version 5.3 and earlier versions were developed in Python programming language, while it was developed in C++ starting from version 6 [4]. Since the invention of BitTorrent, it continued to be a strong file sharing protocol, although the intention was to produce a protocol that replaces HTTP protocol and not to produce a full P2P system [8].

When a considerable number of peers in a given P2P network do not contribute efficiently, they introduce a case called free riding. This situation forms a threat to the proper network operations as the peers in any network are expected to participate as a return of using the network's resources. A free rider is a peer that uses P2P network services but doesn't contribute to the network or other peers at an acceptable level. Different studies showed that 70 percent of peers didn't share any files at all, and 25 % provided 99 % of all resources in the network [9]. The free riding problem will lead to an inefficient system that is similar to a client server architecture where the overall load is focused in the non-free riding peers. It is even worse as the regular peer will not have the capacity and the computation power of a high-end server [7].

In P2P content sharing systems, fairness is considered one of the most important factors of the system efficiency as it encourages peers to participate and share their resources. BitTorrent is not a fair system as it only remembers a short history of the peers upload and download rate. This will be a problem for a system with a homogeneous bandwidth distribution. This is mainly due to the tit-for-tat strategy that is implemented in BitTorrent choking algorithm. In addition, peers will continue to upload and download from the same peers as the tit-for-tat is backward-looking algorithms and not forward-looking. The tit-for-tat is not efficient when a peer wants to collaborate but does not have the piece that meets other peers' interest [8].

Many studies such as [13, 11, 21] showed that free riding is achievable in BitTorrent. Even if free riding did not change the performance of the network, it is generally not fair to have active free riders in any P2P systems. Free riders availability will add an overhead on the other providers in the network. In this case, free riders will consume the resources without the need to give a single piece.

There are many ways free riders can behave in BitTorrent environment. One of the ways is by not sending a HAVE message, in this case other peers in the network will not know that the free rider got a piece and they will not send a request to get that piece. Free riders also will not honestly announce their file status while sending a BITFIELD message while handshaking with other peers. Free riders can exploit some aspects of the BitTorrent Protocol to maximize their download and reduce their upload which degraded the overall performance of the system.

BitTorrent is robust and efficient when used ideally, that is when resources are shared by all the active peers in the file sharing session. However, it can be further improved to enforce fairness especially when free riders peers are active in the network. Depending on how a free rider peer behaves, the solution to mitigate their behavior can be designed and this introduces a variety of research topics. Examples of malicious acting is free riding, uploading junk data, changing open connections configuration, and sending multiple messages to the tracker to get more information about seeders. Other examples of BitTorrent open research areas are peers selection algorithm, next piece selection algorithm and BitTorrent network management [15, 16].

Some studies classified the solutions to mitigate free riders into three main categories. First: monetary based approach which requires peer charging system for the service they get. The fee usually is in a form of virtual currency [22]. Second: reciprocity approach, which uses information about the peers' behavior and their contribution to the environment. Third: reputation based approach, which is based on the reputation of the peers in the system. Each one of these three categories has its advantages and drawbacks. There is usually a trade-off between efficiency and effectiveness in detecting free riders. The problem is even more complicated when individual free riding peers know technical details about the communication protocols and the free riding detections techniques. In this case, they will always have a way to trick the system and get advantage of it [7].

In this work, the free riding problem is addressed through modifying the BitTorrent choking algorithm to overcome the fact that free riding can be achieved via not sending HAVE message when the download of a piece is completed.

## 1.1 Problem Statement

Free riders in this work are the peers who never send a HAVE message when they complete downloading a piece. They also never state their file status honestly as they will always claim that they do not have any piece yet by sending an empty BITFIELD message. In this way, free riders will never share any block in the BitTorrent environment.

It is proven that the existing BitTorrent choking algorithm suffers from many drawbacks. One major drawback is allowing free riders to utilize resources of the contributing peers which will lead to an overall inefficiency for the contributing peers and the environment. It will also cause an overhead on the pieces providers as they will have to distribute the file for free riders and non-free riders. If a peer is going to eventually get what it is asking for whether it shares or not, there will be no incentive to share resources, all peers will rather be free riders as well. In this research, we aim to raise the fairness of the BitTorrent environment by discouraging free riders to join an active BitTorrent file sharing session or even encourage them to change their behavior and start sharing.

By solving the problem of free riding in BitTorrent environment, the load on contributing seeders and leechers will be reduced as all peers in the system will be sharing resources. It will also reduce the load on the tracker as the initial provided peer sets will be sufficient to provide the full file to the swarm. In addition, the download progress of free riders will be slow down as a way of punishment.

It has been proven in the literature and through our experimental results that BitTorrent choking algorithm suffers from many drawbacks. In this research, we try to block some of the holes that free riders use to get advantage of the contributing peers.

## 1.2 Research Objectives

The main objectives of this research are:

- To fully analyze the BitTorrent choking algorithm and find out where free riders are getting advantage.
- To introduce a new type of free riding by not sending a HAVE message when a free rider completes the download of any piece.
- To analyze the effect of free riding in the overall environment.
- To reduce the effect of free riding by improving the existing choking algorithm (tit-for-tat) that is a game theory based model in BitTorrent environment and enhance the optimistic unchoking algorithm to overcome free riders behavior.
- To analyze the effectiveness of the proposed free riding mitigation algorithm and compare the results with the original choking algorithm.

## 1.3 Thesis Contribution

The main contribution of this research is the full analysis of the choking algorithm every 10 seconds and the optimistic unchoking as well. The analysis was done experimentally, to detect the contribution of each block of the algorithm in unchoking free riders. Also the free riding idea which is not to send a HAVE message is unique and original. In addition, implement the X chance giving algorithm that shows an effective way of

delaying the download progress of free riders while maintaining the download progress of non-free riding peers. In addition, the implemented algorithm cannot be tricked by other peers; whether they are honest, dishonest, forms a group or cheat by any other known mean.

## 1.4 Thesis Organization:

The structure of the thesis is organized as the following: chapter 2 gives a background about BitTorrent key terms and definitions, the BitTorrent communication protocol and it also covers some important techniques that BitTorrent uses for improving file sharing efficiency.

Chapter 3 presents the literature review of free riding in BitTorrent and some of the implemented mechanisms to mitigate free riding in BitTorrent environment.

In chapter 4, the thesis discusses the proposed model, where we show the research approach and the modification of the choking algorithm that is the heart of this research.

Chapter 5 discusses the simulation and experimental results. It covers the effect of injecting free riders in the system, the effect of free riders on peers with different bandwidths and the competition between free riders and non-free riders in different simulation experiments. It also shows the improvement of the BitTorrent environment after improving the choking algorithm.

The last chapter of this thesis is a conclusion and a discussion about the future work.

# CHAPTER 2

# BitTorrent Background

## 2.1 BitTorrent Environment Terminologies

Before describing the BitTorrent protocol in details, it is necessary to define the most important terms in BitTorrent environment. Below are the fundamental BitTorrent terms and keywords:

- **.torrent File:** A file that contains the meta-data about the file to be shared. It is usually published by the original file source and uploaded to a public repository. It contains data about the file such as the number of pieces that form the file, the file length, the hash code of the file, the tracker address and other related information.
- **Tracker:** A master node (server) that has all information about all active peers in a BitTorrent session. It provides a list of peers to be contacted for a file sharing session. Also it tracks the progress of the file being downloaded for all peers and manages the communication among peers.
- **Seeder:** A peer that has the completed file. A peer can be an initial seeder, which means that it has the completed file since the beginning of the file sharing session. A peer can be also transferred to a seeder when it completes the file download during the progress of the file sharing session.

- **Leecher:** A peer that has a part of the shared file at a specific time. Ideally and by default, a leecher will be transformed to a seeder when it completes downloading the file.
- **Piece:** A part of the file. It is divided into a set of blocks or sub-pieces. The piece size can be selected by the original source of the file through a multiple choice options while generating the .torrent file.
- **Block:** A sub-piece, the block is the smallest unit of the shared file; its size is 16 KB.
- **PeerSet:** A set of peers that includes seeders, leechers or a combination of both. The tracker randomly provides the peer set to a peer who joins the file sharing session. The peer set size ranges from 20-80, but the standard and default size is 50 peers.
- **Swarm:** A database at each peer, it saves information about all peers in its peer set. It is mainly used to store the file status of each peer (what are the pieces that a given peer has)**.** The data structure of the swarm is a two dimensional array with the peer ID as an index and one dimensional array of 0 and 1 to identify the available pieces with each peer at any given time.
- **Neighbor:** A peer with (TCP/IP) direct connection with another peer. It is in its peer-set and swarm. It is randomly provided by the tracker upon a request.
- **Message:** A message that peers and tracker exchange to manage all transaction during the file downloading and uploading process. There are many BitTorrent communication messages, which will be discussed in more details in this chapter.

## 2.2 BitTorrent Communication Protocol

In BitTorrent, the trackers are responsible for helping peers who need to download a file in finding providers of the file. Trackers uses a very simple protocol on top of HTTP, where it receives information about the file to be downloaded, the port of the sender, and the tracker reply with a list of other peers who are participating on uploading or downloading the same file [5].

All logistical problems about file upload and download are handled at the peers' interaction level. A very little amount of information about file download and upload is sent to the tracker for statistical analysis only. Each time a tracker is asked to provide peers for interaction, it provides a random 50 peers. Random peer selection is proved to be robust as many peers selection algorithms results in power law graph which can get segmented after a small amount of mix [5].

In a BitTorrent peer-to-peer interaction, data is sent using TCP protocol where several requests are pending at the same time. To avoid delay problems while sending the pieces, BitTorrent clients divide the pieces into blocks and send 5 requests at a time.

As described in [6], the BitTorrent file exchange mechanism is run as the following:

- The source of the File (original seeder) creates a .torrent file, which contains meta-data about the file. Such data includes: file name, length, number of pieces, the tracker address and hashing information.
- The original file provider posts the .torrent file in a public location or a website.

- Any peer who is interested in downloading the file will contact the tracker to join an active sharing session and gets a list of other peers who might have a part of the file or the complete file, by using the .torrent file.
- The new peer joining the network will contact the list of the provided peers directly to ask for downloading the file.
- When the new peer downloads the file completely, it forms a new source to distribute the file and will be provided to other peers as a file source.

The original file provider has the ability to divide the file into several pieces where he or she can choose the piece size from a given list while using BitTorrent client. The pieces are further divided into multiple blocks, each of size 16KB. The file structure is designed in a way that supports rapid sharing, where the blocks and pieces can be acquired from different providers.

**Figure 1 summarizes the BitTorrent file sharing process:**

1. The client ‘peer A’ retrieves the .torrent file from a public web-server
2. ‘Peer A’ contacts the tracker asking for 50 active peers in the torrent file sharing session.
3. The tracker sends 50 peers that are active during the torrent file sharing session.
4. ‘Peer A’ contacts the peer set provided by the tracker if possible. Some of them are seeders and the others are leechers
5. File exchange starts according to the BitTorrent protocol, which will be explained later in this chapter.

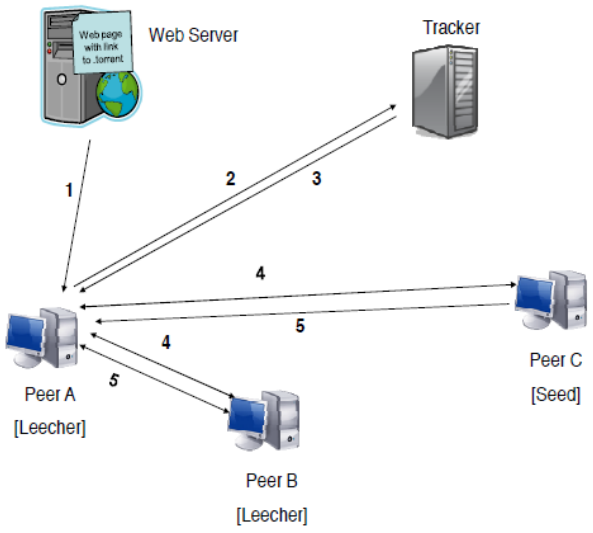


**Figure 1: BitTorrent file sharing process [16]**

Figure 2 shows a snapshot of a BitTorrent client, the upper right corner shows an option of adding a new torrent file; this means that the client wants to download the file. There is also an option on the list to create a new torrent; in this case, the seeder will share a new file and will create a new torrent file to be published at a public website. The lower part of figure 2 shows the status of all the subfolders of the initial folder to be downloaded. There are also other tabs that show information about the peers in direct connection with the client, and information about the bandwidth and the progress of the downloading process.

Figure 3 shows the form of creating a new .torrent file. It gives a list of the piece size, and the user has the option of keeping the default, which will allow BitTorrent to choose the optimum piece size. The user can also choose whether to keep the torrent file private or public. The piece size ranges from 16 KB to 16384 KB.

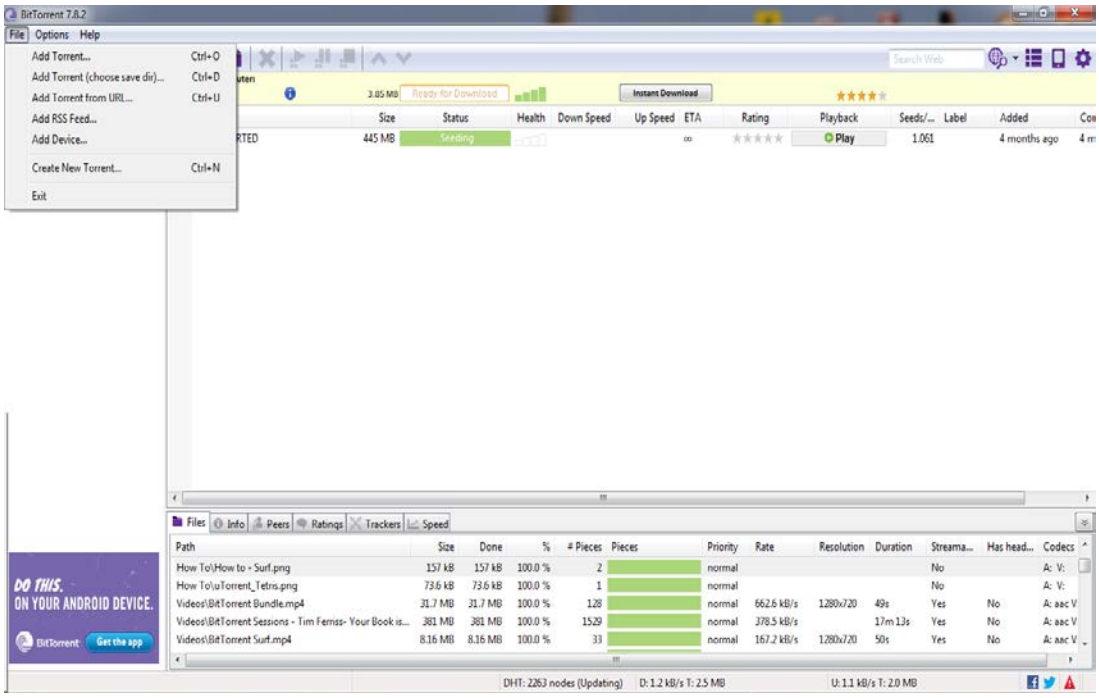


**Figure 2: BitTorrent client**

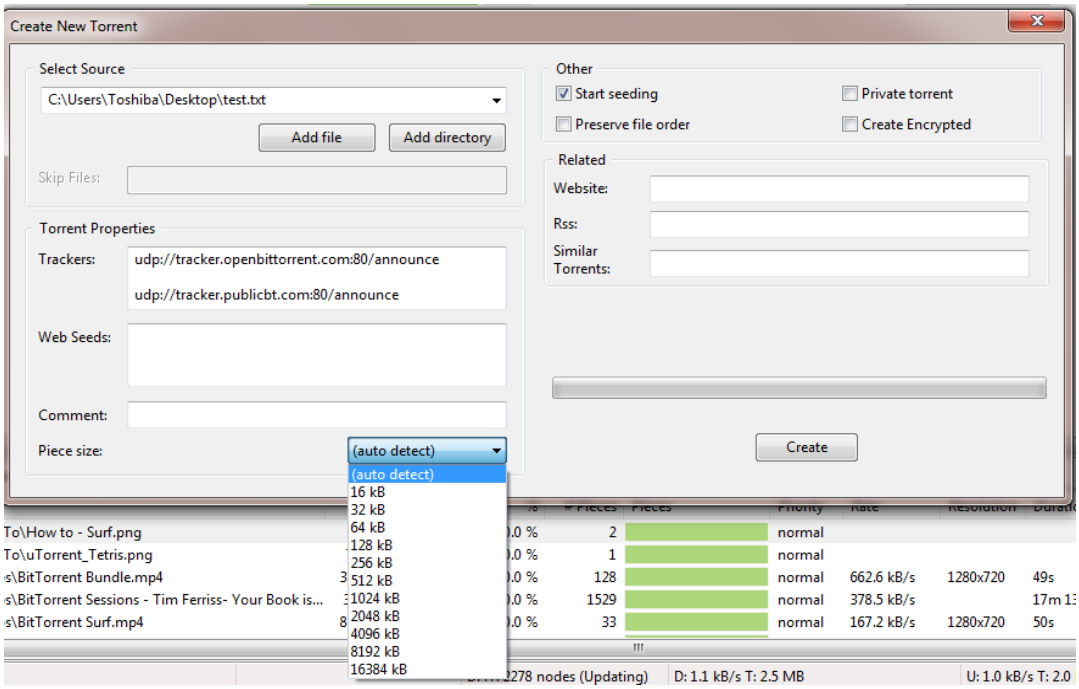


**Figure 3: Creating a new .torrent file in BitTorrent client**

BitTorrent protocol is a message based protocol. These Messages are used to communicate between the peers and the tracker and between the peers with themselves. Below is the list of messages that are used in BitTorrent protocol:

- **TRACKER:** This message is sent from any peer to the tracker to ask for a set of peers that are currently active in either downloading or uploading the file.
- **PEERSET:** A message that sent from the tracker to a peer that asked for a set of peers which are currently active in a file sharing session. The message will contain the addresses of 50 peers.
- **BITFIELD:** A message that is sent between a peer and another peer to become a neighbor. It is also used to establish a direct connection and to share the file status of each peer.

**There are 4 types of BITFIELD messages:**

1. **Request with ACK:** A peer sends the message to another peer provided by the tracker in the peer set. The objective is to ask the remote peer if it can be a neighbor and to ask for its file status.
2. **Response with ACK:** It is used to confirm that the sender of the previous BITFIELD message was added to the swarm of the reception and to confirm that the file status was updated to reflect what the sender has.
3. **Response with NACK:** It is used to inform the sender of a received BITFIELD message of type a **request_with_ACK** that the request is not accepted. This is most probably due to exceeding the number of allowed neighbors. In this case, the swarm and the file status is not updated.

4. **Request with NACK:** This message is sent when a message from an unknown sender is received. This usually happen due to a problem in the transport protocol. The receiver of this message usually sends a **request_with_ACK** to the sender of this message.

- **INTERESTED:** A message that is sent from a peer to another to indicate that it is interested in any of the receiver's resources.
- **NOT_INTERESTED:** A message sent after receiving a HAVE message regarding a specific piece to indicate that the sender is not interested in that piece anymore.
- **CHOKE:** A message that is sent from a peer to another to indicate that it won't allow requesting any pieces from the sender at the moment.
- **UNCHOKE:** A message that is sent from a peer to another to indicate that it is allowed to request pieces from the sender at the moment.
- **REQUEST:** A message that is sent from a peer to another peer requesting a specific block in a specific piece from a specific peer, it can be sent only after receiving UNCHOKE message.
- **CANCEL:** a peer sent from a peer to another to indicate that it already got a piece and it is not interested in it anymore.
- **PIECE:** In this message, the block data is streamed from the source to the destination.
- **HAVE:** A message sent from a peer to all other peers in its swarm, it indicates that a new piece is completely downloaded and it is available for sharing.

- **KEEP_ALIVE:** a message sent between neighbors who have not contacted each other for two minutes to check whether they are still connected to the network.

When a new peer joins the network, it may start with zero pieces or partial pieces of the required file. At this stage, it will send a TRACKER Message to the tracker to ask for other peers who are actively sharing the file at the moment. The tracker will reply back with a PEERSET message that includes 50 random peers who are seeders, leechers or a combination of both.

When the local peer receives the PEERSET message, it will contact all 50 peers using BITFIELD message. The BITFILED message is used to announce the file status of each peer where the swarm of the local peer and each remote peer is being updated based on the file status field attached to the BITFIELD message. At this stage, all leechers will send an INTERESTED message to all peers who have a piece of interest; the piece selection is decided based on the rarest first algorithms. When a peer receives an INTERESTED message, it updates its database of interested peers.

Every ten seconds, each peer runs a choking algorithm to decide which 3 peers it should unchoke. By uncocking, a piece provider will allow the interested peer to download it. The UNCHOKE message is sent to 3 interested peers and the rest of the peers will be sent a CHOKE message. The unchoking algorithm will be discussed in section 2.4.

When a peer receives an UNCHOKE message, it sends a REQUEST message to the peer that just sent the UNCHOKE message. When a peer receives a request message, it sends back a PIECE message with the block that was requested.

When a peer complete downloading a given piece, it sends a HAVE message to all peers that are interested with the piece ID attached to it. When a peer receives a HAVE message, it updates its swarm, indicating that a new peer became a provider of this piece. When a peer completes downloading all pieces, it is switched automatically to a seeder of the file.

If a peer receives a HAVE message for a piece that was received from someone else, it will send NOT_INTRESTED message to the sender of the HAVE message, this is to indicate that the sender of the NOT_INTRESTED message is not interested anymore in the specific piece which was interested in earlier. If a peer sent a REQUEST message to another peer then gets the piece from somebody else, it will send CANCEL message to indicate that the piece has been already collected.

Figure 4 shows the main part of the BitTorrent algorithm. Assuming that ‘Peer A’ is a leecher who just got the ‘Peer B’ and ‘Peer C’ in its peer set that was given by the tracker.

1. ‘Peer A’ send a BITFIELD message to both peers, ‘Peer B’ is a seeder and ‘Peer C’ is a leecher. The BITFIELD message has the file status of ‘Peer A’. Since it is a leecher that just joined the environment, it has Zero pieces.
2. ‘Peer B’ and ‘Peer C’ will replay with a BITFIELD message to ‘Peer A’. ‘Peer B’ will indicate that it is a seeder and ‘Peer C’ will indicate that it is a leecher by attaching their file status to the BITFIELD message.
3. Assuming that ‘Peer A’ is interested in a piece from ‘Peer B’, it will send an INTRESTED message and it will mention the piece ID.

4. ‘Peer A’ will wait for UNCHOKE message from ‘Peer B’ and this might take some time as ‘Peer A’ has to be selected through ‘Peer B’ choking algorithm either through the optimistic unchoking or the regular unchoking algorithm.
5. When ‘Peer A’ receives UNCHOKE message, it will send a REQUEST message attaching the piece ID.
6. ‘Peer B’ will send a PIECE message that has the continent of the block that is previously requested.
7. ‘Peer A’ will repeat the cycle from (3-6) until it complete downloading one piece or it receives a CHOKE message.
8. When ‘Peer A’ complete downloading a piece, it will send HAVE message to all neighbors, including ‘Peer C’ to indicate that it just completed the piece and that it is available for sharing.
9. Assuming that ‘Peer C’ is interested in the above piece, it will send an interested message and the cycle will goes from (3-9) between ‘Peer A’ and ‘Peer C’

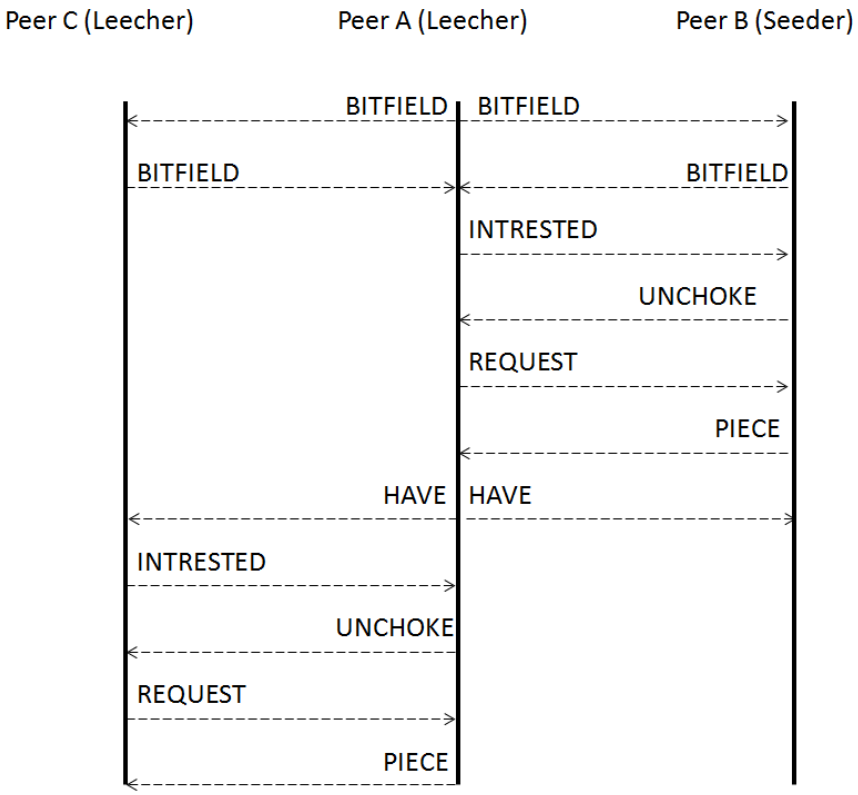


**Figure 4: BitTorrent messages sequence**

## 2.3 BitTorrent Piece Selection Strategies

Piece selection in BitTorrent plays a major role in the performance and robustness of the system. A poor piece selection could lead to having all the pieces which are currently on offer or not having any piece to offer. BitTorrent implements the strict piece policy which means that once a single block is requested, the remaining blocks from the same piece are requested too; this ensures getting complete pieces as soon as possible. BitTorrent also implements the rarest first policy; it decides which piece to be downloaded next. Peers generally start downloading the least available pieces among their neighbors. This means

that the most wanted pieces will be available for downloading in a quick manner. It will simply delay requesting pieces that are commonly distributed for later [16].

There is an exception for the rarest piece selection policy, which is when a new peer joins the network; this is to ensure that the new peer will get a piece as soon as possible, so it can offer at least one piece to the community. For this reason, the new comers will chose the first piece randomly, and then will switch to the rarest piece first policy [5].

There are four main piece selection strategies that peers use in order to select the next piece to download. This section of the thesis will cover the different piece selection strategies in BitTorrent protocol.

- **Strict Policy:** Peers in BitTorrent try to get the complete piece of the file before requesting other pieces. This is due to the fact that only complete pieces can be exchanged. This means that if a block of a specific piece is requested, then all the blocks of the same piece will be requested later [18].
- **Engame Mode:** This strategy is used at the end of the file downloading process. When a peer asks for the same block from all neighbors, and the block is downloaded, it aborts the connection with all the neighbors in order to save bandwidth. This strategy is used to avoid wasting time waiting for slow peers to provide the last block [18].
- **Rarest First:** This strategy means that peers will look for the rarest piece available in the system in order to download it first. The purpose is to reduce the load in the piece provider and have more providers in the system as soon as possible. The peers can identify the rarest piece in the system by searching for

the pieces among all neighbors in the swarm. The swarm is updated every time a piece becomes available in the system [18].

- **Random First Piece:** There is an exception of the rarest piece policy, which is the random first piece. This is adopted when a new peer joins the system as it would like to share a complete piece as soon as possible. This is adopted for the peer to be ready when a peer is evaluated for choking decision using tit-for-tat. This method will choose the first piece randomly [18].

## 2.4 Choking Algorithm in BitTorrent

As BitTorrent is not a central resource allocation system, each peer is responsible for maximizing its download rate. This is achievable by downloading from peers that have the missing blocks and pieces, considering that they are allowed to download; this can be controlled through the choking algorithm. As the adopted choking algorithm is a tit-for-tat, peers have to cooperate in order to maximize their download rate. Choking is a temporary refusal to upload; the choked peer will stop downloading but it can continue upload to a specific peer without the need to reconfigure the connection. The choking algorithm is not a part of the wire protocol, it is only required to enhance the overall network performance. A good choking algorithm should utilize all the available resources. At the same time, it should be consistent resistant to peers that download without uploading resources [5].

In BitTorrent, each peer unchoke a fixed number of peers, the default number is 3. The decision on which four peers to unchoke is taken every 10 seconds by each peer and it is based on the download rate or the upload rate on the last 20 seconds, depending whether

the peer running the algorithm is a seeder or a leecher. This process is repeated every 10 seconds where the three peers to be unchoked are re-picked; this is to avoid wasting resources by rapidly choking and unchoking peers. Previous choking algorithm also consider long term net transfer amounts, this solution performed poorly because of shifting the bandwidth rapidly over time as the resources go away or become available [5].

Choking only peers that upload the most to the local peers will lead to ignoring peers that recently join the network and will also lead to the lack of discovery if the current unused connections are actually better than the used ones or not. To overcome this limitation, BitTorrent uses optimistic unchoking that occurs every 30 seconds by each peer. During this time, one random peer is being unchoked. During optimistic unchoking, the current download rate is completely ignored; this will allow the new comers to download resources. 30 seconds was selected carefully to ensure upload full capacity, download full capacity and for the downloader to reciprocate [5].

The choking algorithm every 10 seconds consists of 4 different blocks. In the first block, there are two scenarios. If the local peer is a seeder, it will sort its neighbors by how much they have downloaded from it in the last 20 seconds. Then it will try to pick the top 3 interested peers, in terms of their download rate. If there are no 3 interested peers, then the second block of the algorithm will be activated.

The second scenario is when the local peer running the algorithm is a leecher. It will sort its neighbors by how much they have uploaded to the local peer and will pick the three

top interested peers, if any. If there are no 3 interested peers, then the second block of the algorithm will be activated.

In the second block of the algorithm, the three slots to be unchoked are complemented randomly.

- If 3 slots were filled by block number 1 of the code, then there will be no slots that are filled randomly.
- If 2 slots are filled by sorting, then 1 slot will be filled randomly.
- If 1 slot filled by sorting, then 2 slots will be filled randomly.
- If 0 slots are filled by sorting, then 3 slots will be filled randomly.

In block 3 of the algorithm, the 3 slots whether filled by sorting or filled randomly or a combination of both will be sent an UNCHOKE message if they have contributed anything in the last 60 seconds and they have contributed anything before, if the local peer is a leecher. If the local peer is a seeder, it will definitely send 3 unchoke message to the peers occupying the three slots.

In the fourth block of the algorithm, all peers that were not unchoked in block 3 will be choked and sent CHOKE message. Figure 5 describes the BitTorrent choking algorithm that happens every 10 seconds by all peers. The numbers in the boxes are the number of blocks uploaded or downloaded by the peers running the algorithm in the last 20 seconds, depending whether the local peer who is running the algorithm is a seeder or a leecher. The pseudo code of the regular choking algorithm for seeders is illustrated in algorithm 1 and the pseudo code of the regular choking algorithm for leechers is illustrated in Algorithm 2.

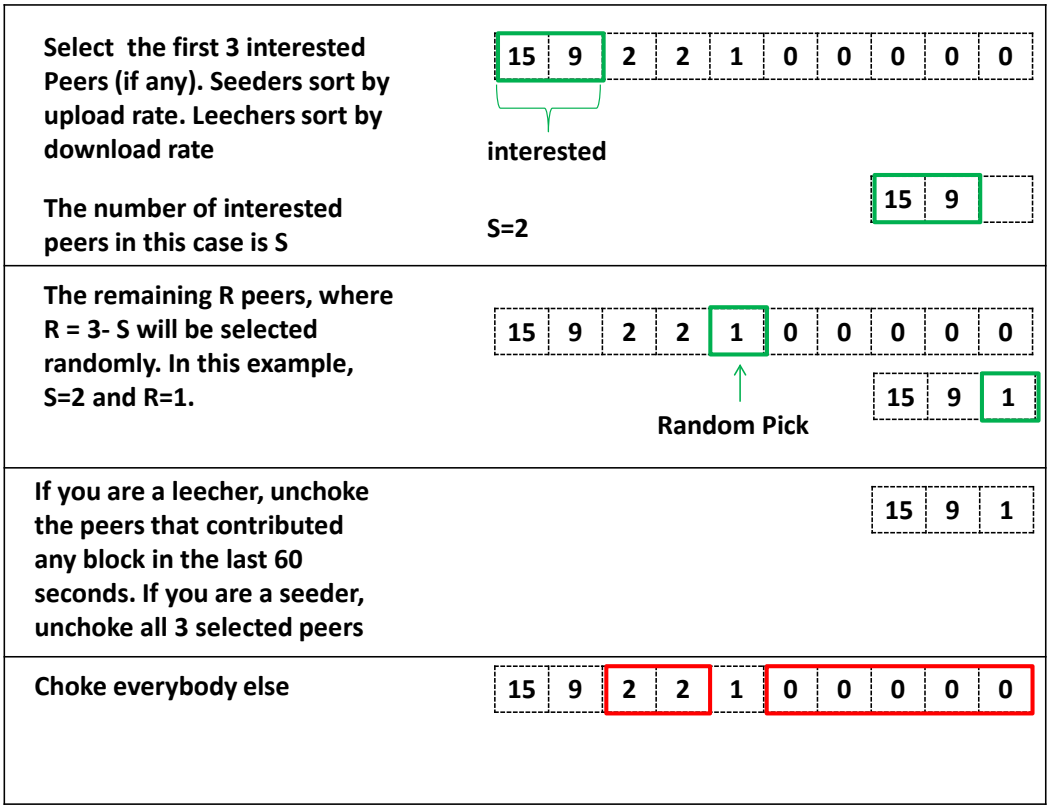


**Figure 5: The four blocks of the original BitTorrent regular choking algorithm**

**Input:** All neighbors of the local peer

**Objective:** Unchoke 3 neighbors and choke the remaining peers in the neighbors list

**// picking interested peers**

*UnchokingCandidates*[]= []

**for** every neighbor in the neighbors list **{**

**If (**neighbor is interested in a piece**){**

*UnchokingCandidates* = *UnchokingCandidates*+ neighbor

**{**

**}**

**//sorting interested peers**

**Sort** *UnchokingCandidates* by their download rate from the local peer in the last 20 seconds in a descending order

**// complimenting interested peers randomly**

**If (***UnchokingCandidates. Length <3***){**

**Fill** *UnchokingCandidates* list randomly until it reaches a total length of 3

**}**

**// performing the chokes and the unchokes**

**Unchoke** the top three peers in *UnchokingCandidates*

**Choke** every other peer in the neighbors list

**Algorithm 1: Seeders regular choking algorithm**

**Input:** All neighbors of the local-peer

**Objective:** Unchoke 0-3 neighbors and choke the remaining peers in the neighbors list

**// picking interested peers**

*UnchokingCandidates*[]=[]

**for** every neighbor in the neighbors list **{**

**If (**neighbor is interested in a piece**){**

*UnchokingCandidates = UnchokingCandidates*+ neighbor

**{**

**}**

**//sorting interested peers**

**Sort** *UnchokingCandidates* by their upload rate to the local-peer in the last 20 seconds in a descending order

**// complimenting interested peers randomly**

**If (***UnchokingCandidates. Length <3***){**

**Fill** *UnchokingCandidates* list randomly until it reaches a total length of 3

**}**

**// performing the chokes and the unchokes**

**for** every Peer in the *UnchokingCandidate* list **{**

**If (**local-peer is not snubbed by Peer **){**

**Unchoke** Peer

**{**

**else{**

**Choke** Peer

**}**

**}**

**Choke** every other peer in the neighbors list

**Algorithm 2: Leecher regular choking algorithm**

Every 30 seconds, all peers will unchoke one peer from their neighbors randomly, this is called optimistic unchoke. The only condition is if they are choked at the time of running the optimistic unchoking algorithm. Figure 6 demonstrates the idea of optimistic unchoking. The numbers in the boxes shows the number of downloaded or uploaded blocks by all the neighbors of the peer running the optimistic unchoking, depending whether the local peer that is running the algorithm is a seeder or a leecher. Algorithm 3 demonstrates the pseudo code of the optimistic choking algorithm for seeders and leechers.

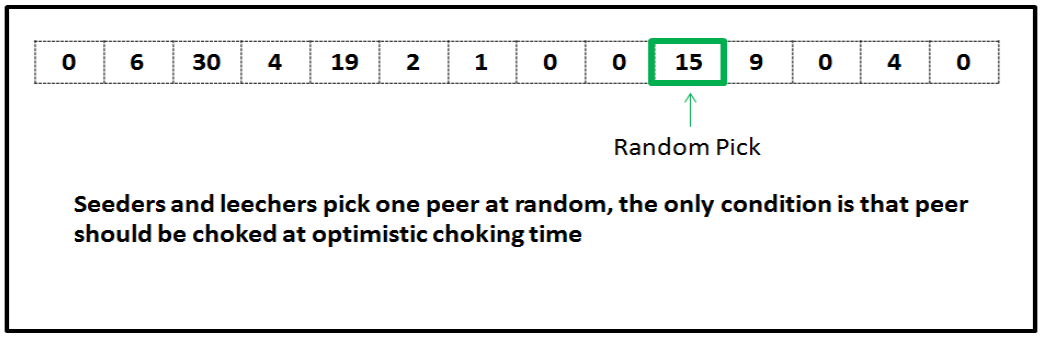


**Figure 6: Optimistic choking demonstration**

```
Input: All neighbors of the local-peer
Objective: Unchoke 1 neighbor and keep the current choking status of the rest

// picking unchoking candidates
UnchokingCandidates[]= []
for every neighbor in the neighbors list {
        If (neighbor is currently choked){
                UnchokingCandidates = UnchokingCandidates+ neighbor
        {
  }
Randomly select one peer from UnchokingCandidates
Unchoke selected Peer
```

**Algorithm 3: Optimistic choking algorithm for seeders and leechers**

Figure 7 summarizes the BitTorrent choking algorithm in a set {n} representing the neighbors of a specific peer that is running the algorithm. The set u is the unchoked peers during the ten seconds interval, while the set o is the unchoked peer during the 30 seconds interval and it is of size one.

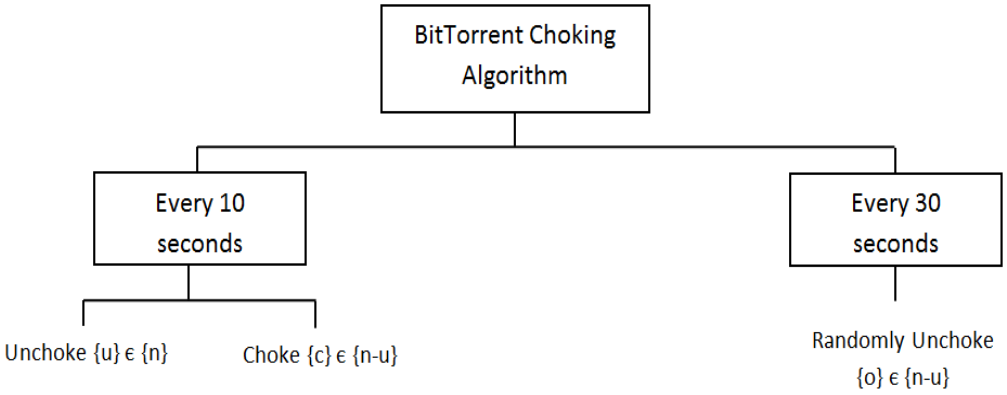


**Figure 7: A summary of BitTorrent choking algorithm**

Below is a summary of the original choking unchoking algorithm:

- Every 10 seconds:
  - If the peer is a seeder, it sorts its neighbors by downloading rate in the last 20 seconds. The top 3 ({u}ϵ{n}) interested peers (if any) in decreasing download rate order will be unchoked.
  - If the peer is a leecher, it sorts its neighbors by uploading rate in the last 20 seconds. The top 3 ({u}ϵ{n}) interested peers (if any) in decreasing upload order will be unchoked.

    - If the number of interested peers is less than 3, the remaining peers to be unchoked will be selected randomly.
    - The remaining peers in {n} that are {n-u} will be choked.
- Every 30 seconds:
    - One peer will be unchoked randomly and this is called optimistic unchoking.
    - Other peers will maintain the current choking status.

The objective of the original design of the choking/ unchoking algorithm differs from seeders to leechers. It is summarized as below:

- Seeder wants to distribute the file as soon as possible and that is why they unchoke the neighbors who downloaded the most out of them in the last 20 seconds of the 10 seconds choking routine. If the number of seeders increased in the environment, the load on the original seeders will decrease as there will be more providers to distribute the file in the environment.
- Leechers want to get the file as soon as possible to become seeder and either leaves the file sharing session or stay as seeder and distribute the file for a specific period of time.
- In optimistic unchoking, both seeders and leechers are randomly picking the peer to unchoke and this is to give a chance to peers with low bandwidth to get advantage of the resources and also to give opportunity to the new peer who just joined the BitTorrent environment.

Considering the described choking algorithm, there are many holes where free riders can get advantages of. Going back to block number one of the choking algorithm, the local peer who is running the algorithm could be a seeder or a leecher. If a seeder sorts by how much it uploaded to its neighbors and unchoke the top 3 peers, then there is no way to detect free riders. Not only this, if the seeder finds that free riders are the fastest downloaders, it will continue unchoking them until they become seeder as well. Then, free riders most probably will leave the environment without contributing a block to the BitTorrent environment. When there are no three interested peers, the remaining will be picked randomly and again there is no way to detect free riders. Figure 8 demonstrates a scenario where free riders can be unchoked by a seeder in the 10 seconds choking algorithm. The numbers in the boxes are the number of block uploaded by the seeder who is running the choking algorithm. ‘FR’ represents a free rider and ‘NFR’ represents a non-free rider.

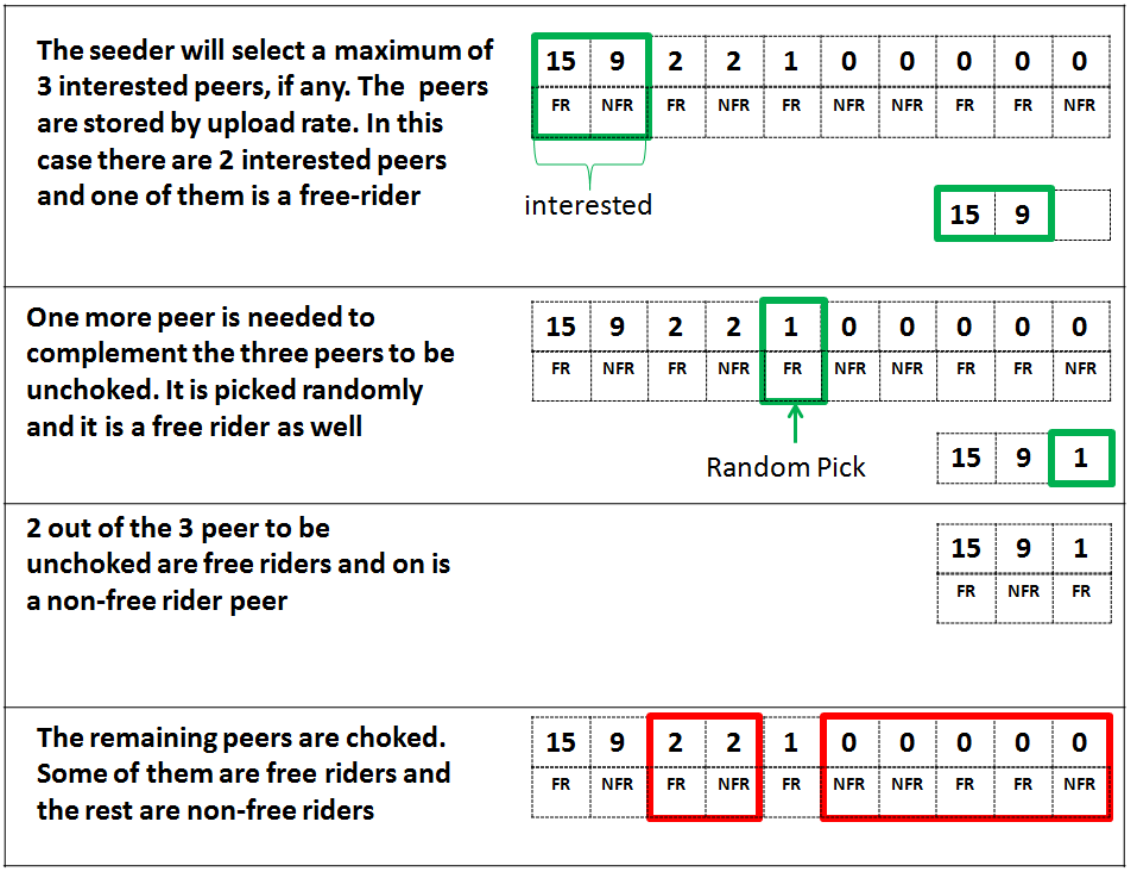


**Figure 8: A demonstration on how free riders get advantages of seeders in the regular choking algorithm**

For leechers, there is more control on free riders. However, there is still an opportunity of a leakage of free riders. When a leecher sorts interested peers by how much they have uploaded to it, zero is still a number and if there are no interested peers that are not free riders, free riders will be selected and unchoked accordingly. It is also the case when there are no 3 interested peers that are not free riders, a maximum of three and a minimum of one free rider will be unchoked randomly. Figure 9 shows one scenario where leechers can unchoke free riders.

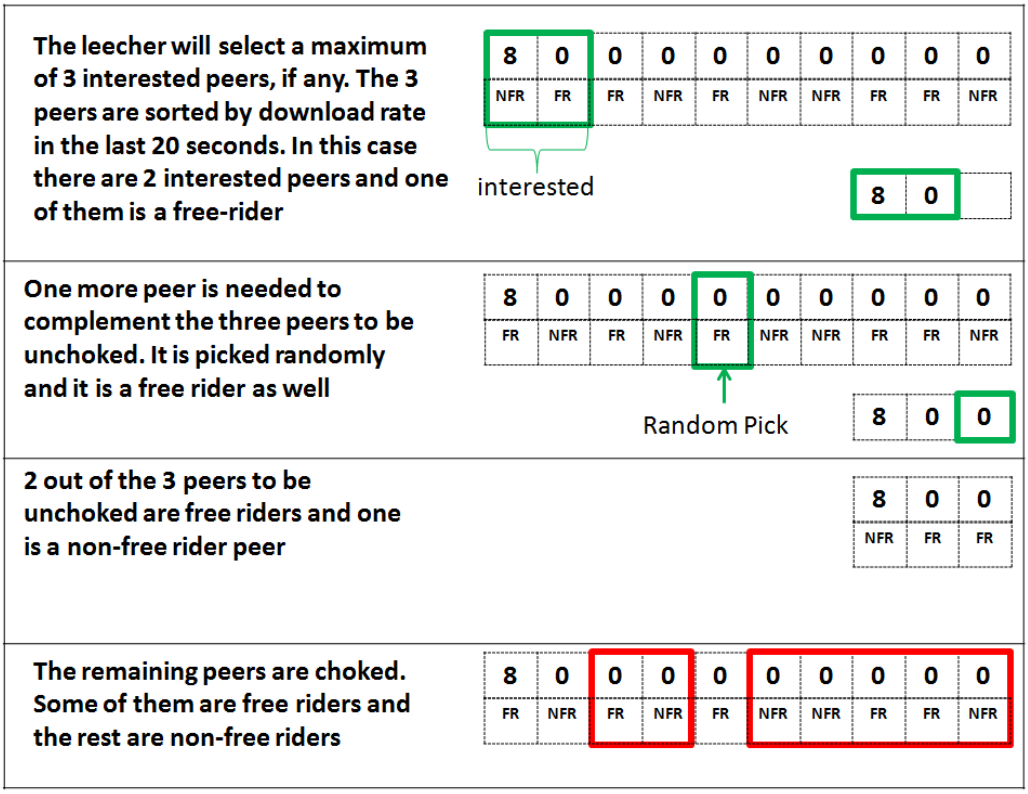


**Figure 9: A demonstration of how free riders get advantage of leechers in the regular choking algorithm**

# CHAPTER 3

# Literature Review

This chapter presents the literature review and the related work in BitTorrent free riding and free riding mitigation. It also discusses the different possible attacks to BitTorrent and the techniques applied to reduce or illuminate their effect on the BitTorrent environment.

## 3.1 Free Riding in BitTorrent Environment

In [13], the author discussed some of the motivations of free riding in BitTorrent environment, such as the limited upload bandwidth and the illegality of uploading copy righted material in some countries such as Switzerland. The author also tried to enhance the protocol and invent mechanisms to make free riding infeasible by implementing ideas from the network coding approach. The author discussed the limitations of the original choking algorithm that is when a peer has less than predefined number of connections into the swarm, it will unchoke any peer that is interested without even contributing any block. BitThief was also introduced in this work, showing the concepts and ideas and the different attacks implemented.

The first attack implemented in [13] is aggressive connection opening, where BitThief tries to open as many active connections as possible; this is achievable by asking the tracker frequently about peer addresses. The experiments showed that after six minutes, the BitThief was able to open three times the connections opened by the official mainline

client. The official client default number of connections limit is 100 connections, while BitThief does not impose a limit in the number of open connections; it aggressively tries to open as many connections as possible. Figure10 shows a comparison between the number of connections BitThif can open and the official client possible number of connections with the progress of time. As the number of connections increases, the download rate increases as well [13].

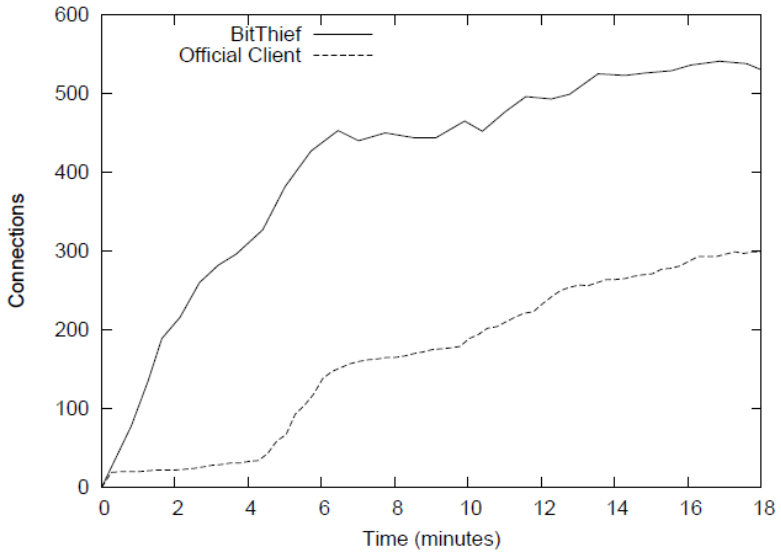


**Figure 10: BitThief and official BitTorrent client number of connections comparison [13]**

The second attack discussed in [13] is called ‘downloading from seeders only’. The reason to attack seeders is the fact that they have the complete file and they do not have any means to check the contribution of peers before unchoking them. Any torrent file starts with one seeder and many leechers, these leechers become seeders with time; and it is up to them when to quit the session and stop providing. BitThief gets benefit of

downloading from seeders and does not depend on leechers as their contribution does not boost the performance [13].

The third discussed attack in [13] is uploading garbage data. During the uploading process, there is no mechanism to check the validity of a received block until the entire piece is completed. In this attack, random garbage is uploaded to trick the receiver that the sender is actually uploading data. Some BitTorrent implementations were able to detect this attack because they could discover that the data uploaded was garbage, which leads to IP blocking. In addition, an algorithm was developed to make sure that no peer will receive an entirely garbage piece, so they will get some garbage blocks only. This problem is also related to the piece size which ranges from 64KB to 1MB, depending on the choice of the original seeder of the file [13].

In [11], the author presented a fluid based model with two different classes of peers to capture the effect of free riding on BitTorrent environment. The study concluded that the incentive mechanism in BitTorrent is successful in preventing free riding in a system without seeders. However, the mechanism is not as successful as in an environment with a high number of seeders. As a result, they proposed a method that limits the seeders bandwidth allocation to reduce the effect of seeders in free riding. It is also found that the weakest point in BitTorrent is the choking algorithms, where there is no effective policy on whom to unchoke next. The experiments performed in [11] concluded that the Nash equilibrium of the system is to achieve the maximum uploading bandwidth. The results show that the algorithm effectively enhances the performance of peers' contribution.

The author of [12] indicates that BitTorrent protocol for large file distribution strives to provide a form of fairness where clients who don't contribute to the environment should not achieve high download rate. In this work, they studied the effect of selfish BitTorrent clients; where they attempt to download more than the fair rate or share; they do that by abusing the protocol mechanism. Three potential exploits where selfish peers can increase their gain were identified. The three exploits were selected after a deep understanding of the BitTorrent mechanism.

The first exploit that was experimented is downloading from the seeders only, this occurs by continuously asking for new peer set by sending a TRACKER message again and again. In a short time, this peer will have information about most of the seeders in the system as it can determine which peers have what pieces through the BITFIELD message. This way, the selfish peer will ignore leechers and contact seeders only. Experiments confirmed that the selfish peer would be able to achieve a high download rate as it is utilizing seeders that usually have high upload rate. In a short time, the selfish peer will have the complete file without the need to contribute any bit in the network. These results violate the fairness model of BitTorrent, according to which free riders should achieve a low download rate only. It will also negatively affect the non-free rider peers in the system, as the selfish peer will utilize one of the unchoking slots of the seeders. The negative effect can be even magnified using Sybil attack through having multiple identities for the selfish peer that maintains multiple open connections [12].

The second discussed attack in [12] is downloading from the fastest peers. This way, the selfish peers will attempt to maximize their download rate by connecting with the fastest peers in the network. This can be achieved through the optimistic unchoking where all

peers can download from any other peer even if their bandwidth capacity doesn't match. The way to detect fast peers is by monitoring their frequencies of sending have message that indicates the complete download of a new piece. This can be applied only to leechers as seeder will not send a HAVE message. This attack is harmful especially during startup phase where new arrivals can only get pieces through optimistic unchoking [12].

The third studied attack is advertising false pieces, which means that a selfish peer can upload garbage when it is asked about a piece. The receiver will discover that he got garbage when he receives the remaining blocks of the garbage piece, this is because it will not match the hash coding of the meta file and there is no way to go back to the origin of the piece, so that there is no way to detect the selfish peer. The selfish peer will generate the garbage data at a very slow rate, so that it will not be marked as a seeder in the system [12].

The author in [19] also developed a BitTorrent client that acquires more than normal view about the available peers in the session and connects to larger number of peers. Also, the modified client will never upload any piece to the environment. The modified client was able to achieve a better download rate, comparing with the official BitTorrent client. The experiment using planetLab residing torrent with 300 leechers shows that free riding peers performed better than the tit-for-tat complaint peers, especially when the free riders are not dominating the system. The modified client connects to the tracker every 15seconds and every time it acts like a new peer, but it keeps the previous information about the other peers it connects to. This way, it can connect to a larger swarm and get more resources from the environment. In this scenario, the client can change identity, not

to be detected by the tracker or the different peers that exchange peers info in case of tracker frailer.

The author in [14] also discussed many of the BitTorrent possible free riding attacks. These include the garbage uploading and the Sybil attack, where clients use different identities to connect to the tracker; this way, they can get more connection to peers that they can download from. The paper also discussed a background about BitThief and the fact that it maximizes its download mainly from seeders and only gets from leechers if possible. BitThief connects to a large number of peers to get a larger opportunity of being unchoked through optimistic unchoking. The author also discussed BitTyrant client, where it will upload in a minimal speed that will allow it to get through its neighbors active peer set. In this case, the BitTyrent was able to achieve 70% improvement in the download rate comparing with honest peers.

Another attack discussed in [14] is the under reporting strategy where a peer does not announce some pieces that it has in order not to share them at the early stages of the sharing session. This will trick the piece selection algorithm to make the client as a less interesting peer of the swarm. If all clients use this methodology, the performance of the entire system will be reduced sharply.

## 3.2 Mitigating Free Riding in BitTorrent Environment

This section discusses the most important free riding mitigation solutions in BitTorrent. Generally, any of the below four solutions could be used in mitigating free riders in BitTorrent environment.

**Monetary-Based Approaches:** This solution is based on applying a charge for the services that peers consume in the network. This charge can be either by applying subscription fee or fee per transaction. There are many problems with this solution, such as the lack of peers' participation incentive and the lack of a guaranteed quality of the service provided. This solution is also based on a centralized monitoring mechanism which might cause an overhead as well as forms a single point of failure for the entire system. It also requires creating an identity system to identify peers.

**Reciprocity-Based Approaches:** This solution is based on monitoring other peers' behavior through a numerical analysis. In this approach, peers usually monitor others behavior during the current session. Since this approach does not keep a historical record of the peer's behavior, a specific peer might be considered as a free rider in one transaction and not in other transactions. In this approach, the peer is being judged based on its direct experience with other peers.

**Reputation-Based Approaches:** This solution is based on the information gathered from all peers in the system. The feedback could be positive or negative. The positive feedback is usually given about good peers while negative feedback is given about free riders. This approach is based on fixed peers identity. This solution has many drawbacks such as collusion, where group of free riders collaborate to give an individual free rider or a set of them a good reputation to damage the reputation system. Another problem is whitewashing where a peer changes its behavior in order to gain the trust of other peers in the system.

**Game Theory Approaches:** This solution is based on game theory which is a strategic approach based on mathematical model to develop a decision making mechanism and approach for each member in a game. In this case, each peer in BitTorrent environment will take a decision to share or not to share based on the information available to it about the other peer that is interacting with. The original solution in BitTorrent is tit-for-tat, that is why the BitTorrent choking algorithm is considered a game theory approach.

Some experiments performed in [18] proved that free riding is achievable in BitTorrent. He observed that an overhead of up to 430% easily generated by a small set of selfish users. To mitigate the effect of free riders in the system, a new P2P architecture was proposed to classify peers with the help of a small number of trusted auditors (TAs). TAs are regular clients that can download and upload resources. However, they can detect free riders by observing their neighbors behavior, this is to improve performance and distribute computation over different nodes. The work of [18] shows that an increase in bandwidth capacity for 80% of free riding identities in a single TA system with a 6% download time while it reaches 100% overhead without the TA implementation.

In this approach, the author of [18] found that the performance of honest peers in BitTorrent mainly suffers from two factors. First, the randomness of selecting peers by the tracker when it is asked about a peer set. In this factor, free riders consume much of the seeds' upload and download rate, while honest peers get lower rates. Second factor is optimistic unchoking where it is very slow in helping honest peers to discover each other.

The proposed approach separates the service pool for the discovered free riders from the service pool of honest peers. The trusted auditor will be able to identify free riders based on their bandwidth analysis and level of contribution. TAs are managed by the content provider and they are controlled by strong identity control through passwords. The proposed approach will not suffer from the reputation collision as there will be no way for a malicious peer to miss with the reputation system with incorrect information about peers' behavior as the reputation system itself is distributed and tightly protected.

In the basic implementation of the proposed methodology, three service pools were created. One pool for the contributing peers, one pool for the free riders and one pool for new comers in the environment. The TA of a specific SWARM will submit the statistics to the tracker and the tracker can then move a peer to a specific pool based on its analyzed reputation. The number of TAs in the system and the frequency of assigning pools to peers are controlled by parameters to the simulation environment. The reputation is tied to the peer identity, this is to ensure history association, all clients request to download a file will have to login to the system with a user name and a password [18].

The author of [10] discussed how incentives play a major role in BitTorrent in motivating peers to upload to others in order to achieve a high upload time for all peers. The author claims that BitTorrent does not provide the incentives to follow the protocol. In his work, [10] proposed an auction-based model to study and improve BitTorrent incentives. Game theory was used to model the new proposed interaction algorithm called proportional share. Studies show that BitTorrent incentives have focused on how to model the unchoking algorithm that peers use to decide on whom to upload to. In this work, the author also focused on developing new and robust incentives by revealing a rather

straightforward model in the way peers clears their action by deciding how much they reward others for uploading pieces to them.

Other work has also been done in encouraging cooperation among selfish BitTorrent participants. tit-for-tat for example is a common mechanism in which peers share resources with peers that cooperated with them in the past. Second example of incentive based solution in BitTorrent is Monetary Mechanism such as BitStore and Dondelion which are centralized approaches that gives incentive to seeders who recently completed the file to remain in the system and continue uploading files; this address the seeder promotion problem. Third example is Topology-based Reciprocation, which allows peers to punish nodes both upstream and downstream from it. In this scenario, peers have to take exactly as much they receive which is unnecessary strict [10].

Regarding the upload garbage attack, the metafile file contains SHA1 hashes for each piece. As a result, the piece can be verified when it is completely downloaded. The peers try to get the entire piece from the same source, which let them discover the validity of the piece in a quick manner. If a peer is discovered uploading garbage data, its IP will be blocked for several hours or days, depending on the implementation. This concluded that this attack is not a preferable option as most of implemented clients such as Azureus can handle it. In addition, there were some attempts to create a hash tree at the sub piece level that allow it to discover the garbage data immediately [13, 14].

There were also some attempts that are discussed in [19] to improve the optimistic unchoking algorithm, so instead of randomly picking peers to be unchoked, the local peer

considers many factors before unchoking a peer. Some of the factors that can be considered while performing an optimistic unchoking to mitigate free riders are:

1. Claim to have no pieces.
2. Claim to have pieces that the local peer is missing.
3. The remote peer is choked for the longest period of time.
4. The remote peer has uploaded the most to the local peer in the past.

Regarding the Sybil attack that increases the chance of being unchoked by entering to the system with different identities and contacting the tracker with different connections, there were many attempts to solve this problem by not allowing the peer to setup more than one connection. One example of client's implementations that prevents such attack is Azureus [14].

The author of [12] concluded the work by discussing the patterns that contribute to BitTorrent robustness. First, is the ability to maintain multiple parallel connections with multiple peers at a time. There is a solution to the first exploit, which is "Super seeding" policy; this is achieved by masquerading as leechers and gradually advertising available pieces. To solve the third problem that is uploading garbage, it is required to have a memory of past interaction to remember the origin of garbage pieces and possibly punish the free riders. The trade-off this solution is the impact on performance since all history has to be saved and checked for each interaction. It is also important to mention that the principle of problem partitioning should be strictly enforced; this means that selfish clients should never be able to influence good clients by distributing false information. One solution to the second exploit that is downloading from fast peers only is to hide

some information about peers; an example is to hide whether a certain peer is a seeder or a leecher [12].

The author of [1] also discussed a way of mitigating free riders in BitTorrent. The method is based on encrypting the exchanged pieces and the decryption is only sent when the other peer uploads some pieces. This is only applicable between leechers as seeders do not require any pieces to download.

# CHAPTER 4

# Proposed Model

## 4.1 Overview

The proposed methodology was implemented using a modified BitTorrent protocol, where free riders are labeled based on a given percentage. In this case, free riders never send a HAVE message and whenever they are asked about their file status; they always claim that they do not have any piece to offer.

The proposed solution is based on improving the original choking algorithm in the optimistic and the regular choking algorithm. We have focused on improving the weakness of the algorithms; it is where free riders take advantage of the initial algorithm, that is when peers get unchoked without considering their contribution to the environment or checking whether they are potential free riders or not. The solution is based on X chance giving algorithm that we have developed. The algorithm analyzes the behavior of each peer before it gets unchoked by monitoring its contribution starting from the beginning of the simulation of is the file sharing session.

The suggested solution was experimented with different network sizes, different percentages of free riders and different percentages of initial seeders. The effect of the free riders and the proposed solution was monitored accordingly.

The below research approach has been adopted in order to meet the research objectives:

1. Analyze the original BitTorrent simulation results without introducing any changes. The objective is to understand how the simulator works and how to change the different configuration parameters and analyzing their effect in the simulation behavior.
2. Introduce the concept of free riding. Free riders will not send a HAVE message and will always claim their file status to be empty, which means that they have zero pieces. Free riders are labeled randomly following a predefined percentage of the whole network size. The only condition to label a fee rider is not to be a seeder.
3. Monitor the simulation behavior after introducing the concept of free riding with different percentages of free riders within the whole population of peers. The objective is to analyze the system behavior after introducing free riders, and to find out if the system behavior is consistent with the theoretical expectations of introducing free riders in the BitTorrent environment.
4. Make sure that free riders do not contribute any block through monitoring the contribution of each peer. Also make sure that non-free riding peers never send a REQUEST message to free riders as non-free rider peers never realized that free riders actually downloaded any piece.
5. Analytically study the choking algorithm in every 10 seconds and optimistic unchoking every 30 seconds. The objective is to investigate where free riders can be unchoked and what kinds of peers unchoke free riders the most.

6. Statistically and analytically find the weaknesses of all choking algorithms and find where free riders are getting advantages of the resources the most.
7. Design a modified choking algorithm with the objective to mitigate and delay the progress of free riders download activity.
8. Modify the original optimistic choking algorithm to mitigate free riders and reduce their effect on the environment.
9. Monitor the improvement in the overall simulation results and measure the improvement on the reduction of free rider utilization of the overall resources.
10. Modify the 10 seconds choking algorithm to mitigate free riders and reduce their effect on the resources of the BitTorrent environment using the same logic implemented at the optimistic unchoking routine.
11. Analyze the results and measure the improvement after implementing the changes in the optimistic unchoking and the 10 seconds unchoking routine.
12. Modify the simulation parameters such as the percentage of free riders, the percentage of initial seeder and the number of peers, then analyze their effect in the simulation behavior.

## 4.2 Modifications on the Choking Algorithm

Let us recap on the weakness of the original BitTorrent choking algorithm. The weakness differs between seeders and leechers. For seeders and during the choking or unchoking decision, they sort their neighbors by how much they have uploaded to them and they pick the top 3 neighbors. This means that there is no control on whether these neighbors are free riders or not. In fact, if they are free riders, they will continue to take and take as

they will be the 3 peers who have downloaded the most in every choking cycle. If there are no 3 peers that are interested, the other will be picked randomly and here is another chance for free riders to get unchoked.

For leechers, during choking decision making process, they sort their neighbors by how much the neighbor uploaded to the local peer in the last 20 seconds. Therefore, there is more control in the free riders. However, ZERO is a number and if there is an interested free rider with zero uploads to the local peer in the last 20 seconds, it will be picked. If there are no 3 peers that are interested, the other will be picked randomly and here is another chance for free riders to get unchoked

In the optimistic unchoking, it is purely random for both seeders and leechers and this forms a chance for free riders to be unchoked.

In any improvement to the original choking algorithm, the original objectives of BitTorrent must be maintained. The original objectives of the choking algorithm are:

- Seeders want to distribute the file faster, and this is why they sort their neighbors by how much they have uploaded to them before. This is to make them seeders as soon as possible. The aim here is to reduce the load on the seeders by producing more seeders in the environment.
- Leechers want to get the file as soon as possible, and this is why they sort their neighbors by how much they have downloaded from them before. This is tit-for_tat.
- The optimistic unchoking is designed to give opportunity to new peers who join the environment because they will not be among the top 3 peers whether they are

sorted by seeders or leechers. Optimistic unchoking also gives a chance to leechers to get introduced to new connections which might be better than the current ones. This way, they will download the file faster.

Our modified choking algorithm maintains the three objectives mentioned above and also mitigates free riders by slowing down there download rate by reducing the number of UNCHOKE messages they receive. We kept the four blocks of the choking algorithm as illustrated earlier; this is to maintain the objectives of the algorithm executed every 10 seconds by all peers. The modification is implemented in the third block (block C) of the choking algorithm that occurs every 10 seconds. In block C, which is the third block of the algorithm, the local peer send the UNCHOKE message to the three peers who selected by sorting or randomly. Figure 11 demonstrates the block where our modified algorithm takes place.

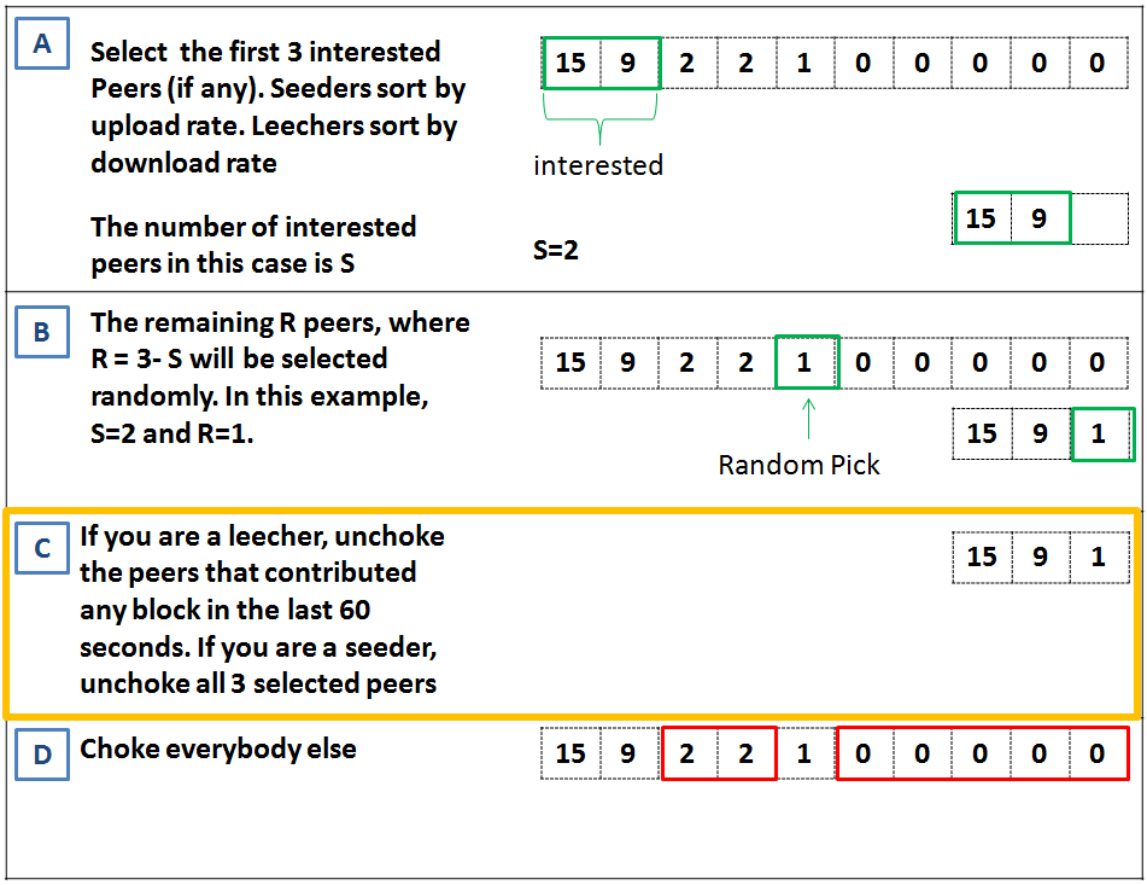


**Figure 11: The block where the modified regular choking algorithm takes place**

The idea of the regular (every 10 seconds) modified choking algorithm is as follows:

Leechers will unchoke the peer (remote peer) occupying one of the three slots selected either by sorting or by random if it meets the below conditions:

- The remote peer did not download at least X amount of blocks from the local peer.
- The remote peer has downloaded at least X blocks from the remote peers and contributed at least on block to the local peer since the beginning of the simulation.

If the remote peer has downloaded X blocks from the local peer but have not uploaded anything to the local peer since the beginning of the simulation, it will never be unchoked by the local peer and in this case it is a suspected free rider.

The idea here is to give free X blocks without running any checks on the peers to be unchoked. The objective of these free X blocks is to give a chance to new comers to contribute, also to give chance to peers with low bandwidth to contribute at least one block to the local peer. This is to be unchoked in the coming unchoking cycle. The pseudo code of the modified leechers choking algorithm is demonstrated in algorithm 4.

```
Input: All neighbors of the local-peer

Objective: Unchoke 0-3 neighbors and choke the remaining peers in the neighbors list

// picking interested peers

UnchokingCandidates[]=[]

for every neighbor in the neighbors list {

        If (neighbor is interested in a piece){

                UnchokingCandidates = UnchokingCandidates+ neighbor

        {

  }

//sorting interested peers

Sort UnchokingCandidates by their upload rate to the local-peer in the last 20 seconds in a descending
order

// complimenting interested peers randomly

If (UnchokingCandidates. Length <3){

Fill UnchokingCandidates list randomly until it reaches a total length of 3

}

// performing the chokes and the unchokes

for every Peer in the UnchokingCandidate list {

        If (local-peer is not snubbed by Peer ){

                If (local-peer. Upload to Peer <X){

                Unchoke Peer

                }

                Else if( local-peer. Download from Peer >0){

                        Unchoke Peer

                }

                else{

                        Choke Peer

                }

        }

}

Choke every other peer in the neighbors list
```

**Algorithm 4: Modified leechers regular choking algorithm**

For seeders, there are two cases: First, a seeder who is initially a seeder. This means that the peer started the file sharing session (the beginning of the simulation for our study) as

a seeder. The problem here that there is no experience for the initial seeder in downloading from others and thus, it cannot tell whether the peer to be unchoked is a free rider or not. In this case, the choking algorithm will remain the same as illustrated in algorithm 1.

The second case is when the seeder was initially a leecher and thus has an experience in downloading blocks from others when it was a leecher. In this case, the seeder who used to be a leecher will run the same algorithm for the leecher who is still a leecher, except in the sorting part where seeders still sort by their upload rate to their neighbors. The pseudo code of the modified seeders with experience choking algorithm is demonstrated in algorithm 5.

```
Input: All neighbors of the local-peer

Objective: Unchoke 0-3 neighbors and choke the remaining peers in the neighbors list

// picking interested peers

UnchokingCandidates[] = []

for every neighbor in the neighbors list {

        If (neighbor is interested in a piece){

                UnchokingCandidates = UnchokingCandidates+ neighbor

        {

  }

//sorting interested peers

Sort UnchokingCandidates by their download rate from the local-peer in the last 20 seconds in a
descending order

// complimenting interested peers randomly

If (UnchokingCandidates. Length <3){

Fill UnchokingCandidates list randomly until it reaches a total length of 3

}

// performing the chokes and the unchokes

for every Peer in the UnchokingCandidate list {

        If (local-peer. Upload to Peer <X){

                Unchoke Peer

        }

        Else if( local-peer. Download from Peer >0){

                Unchoke Peer

        }

        else{

                Choke Peer

        }

}


Choke every other peer in the neighbors list
```

**Algorithm 5:Modified experianced seeder regular choking algorithm**

Experiences seeders and leechers also apply the same algorithm in the optimistic unchoking which runs every 30 seconds, while initial seeders run the original optimistic

unchoking algorithms as described in algorithm 3. The pseudo code of the modified optimistic unchoking algorithm is illustrated in algorithm 6.

```
Input: All neighbors of the local-peer

Objective: Unchoke 1 neighbor and keep the current choking status of the rest


// picking unchoking candidates

UnchokingCandidates[]= []

for every neighbor in the neighbors list {

        If (neighbor is currently choked){

                UnchokingCandidates = UnchokingCandidates+ neighbor

        {

  }

*Randomly select one peer from UnchokingCandidates

If (local-peer. Upload to selected-Peer <X){

        Unchoke Peer

}

Else if( local-peer. Download from Peer >0){

        Unchoke Peer

}

Else{

        go back to *

}
```

**Algorithm 6: optimistic unchoking algorithm for leechers and seeders with experience**

# CHAPTER 5

# Simulation and Experimental Results

This chapter of the thesis discusses the simulation results starting with simulation environment, the effect of injecting free riders in the system, unfairness of BitTorrent system, and the results of applying the modified choking algorithm.

## 5.1 Simulation Environment

The adopted simulation environment to achieve the research objectives is a BitTorrent events based simulator, utilizing PeerSim which is a java based full P2P simulation environment. PeerSim is a simulation environment for P2P protocols that provides extreme scalability for a dynamic environment. It is a highly scalable environment that can be customized through simple configuration parameters.

The file structure is split into pieces of size 256 KB, each piece is divided into 16 blocks, each block is 16 KB. There is a total of 390 pieces for the file to be shared.

PeerSim BitTorrent is an event based simulator. All messages between peers and events are queued and scheduled for an execution with a timestamp. In addition to the messages mentioned in the BitTorrent Background chapter, there are other events that are scheduled and inserted in the same simulation queue. Below are the different types of events that are executed while the simulation is running.

- **CHOKE TIME:** An event that occurs every 10 seconds; it is the shortest interval for among all the events. It is scheduled by all nodes and inserted whenever the same event is under execution. The purpose of this event is for every peer to check which peer to choke and unchoke with the choking algorithm.
- **OPTNCHOKE TIME:** It occurs every 30 seconds by all peers; it is for all peers to run their optimistic unchoking algorithm.
- **ANTISNUB TIME:** This event occurs every one minute, and it is used to check if a leecher is snubbed by another peer. That means that the peer running this event did not receive any block from the remote peer in the last 60 seconds and it provided at least one block before. In this case, the remote peer will be choked and it has to wait for the optimistic unchoking to be unchoked.
- **CHECKALIVE TIME:** This event occurs every 2 minutes, and it is used to check if a remote peer form the neighbors list of a local peer is alive or it left the system. Basically, the local peer will send a KEEPALIVE message to all of its neighbors and those who do not respond within 2 minutes will be considered dead.
- **TRACKER ALIVE:** An event that occurs every 30 minutes to check if the tracker is a live or not. If not alive, peers will not send more TRACKER messages.

The simulation environment is highly scalable through a text configuration file. It provides an easy way of changing the simulation environment such as the number of peers in the network, the file size, the percentage of initial seeders, the experiment end-time, the observation frequency and many other parameters to control the environment.

One of the most important parameters is max_swarm_size, that is to specify the swarm size of each peer. The default max swarm size is 80. Another parameter is peerset_size and this is the default number of nodes that will be given to a node by the tracker when a peer joins the session and asks the tracker for a peer set. The default peer set size is 50. Another important parameter is newr_distr and this determines the percentage of new peers (peers with zero pieces) in the system. To illustrate, if newr_distr is set to 10, the percentage of peers with 0 pieces at the beginning of the simulation will be 10%. Similarly, seeder_distr is a parameter that determines the percentage of seeders in the environment. This represents the peers that have the complete file starting from the beginning of the simulation. For our case, the file consists of 390 pieces.

Free riders are labeled in the NetworkInitializer Java class with a percentage that can be changed manually through a parameter. The only condition that must be met is the selected peer to be labeled is a leecher. The BitTorrent protocol is implemented in the BitTorrent.java class. The observer class is used to collect statistics of the simulation at fixed periods of time that can also be predefined in the configuration file.

## 5.2 Injecting Free Riders into the System

As mentioned, the original BitTorrent code in the PeerSim simulator does not cover the concept of free riding at all. This research only covers one sort of free riding in BitTorrent, that is not to send a HAVE message when a free rider completes the download of a specific piece. Free riders also do not honestly state their BITFIELD file status; they always claim that they do not have any piece to offer to the environment.

The code to simulate the free riding behavior as mentioned above is implemented where the percentage of free riders can be defined and free riding peers are labeled randomly. The only condition to label a free rider is not to be a seeder. In this research, we assumed that only leechers could be free riders as there is no gain for a seeder to be a free rider.

Figure 12 shows the effect of injecting different percentages of free riders in the number of overall exchanged messages on a network of 100 peers with 15% initial seeders in the system, which leaves 85% leecher peers. The percentages of defined free riders will be taken from these remaining 85% peers. The experiment was repeated 50 times with the same setup and the average is presented below.

Figure 12 shows a direct relationship between the number of free riders in the system and the number of all kind of messages sent during the entire simulation. There is a huge reduction in the network traffic as a result of not sending the HAVE message. The number of sent messages dropped from around 33 million to around 4.5 million message.

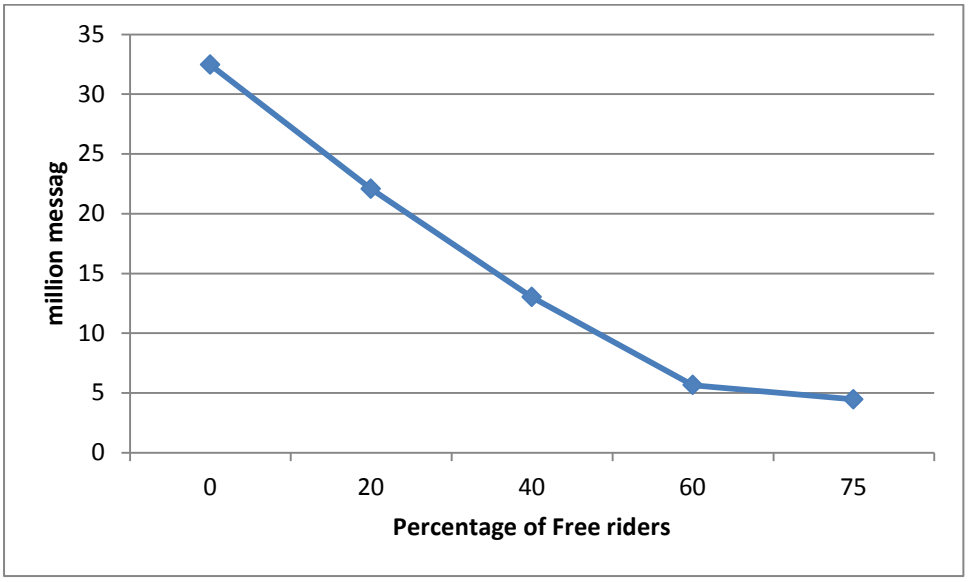


**Figure 12: The effect of Free riders on the number of exchanged messages**

When a free rider does not send a HAVE message, there are a lot of consequences related to that. First, if other peers have not received a HAVE message from the free rider, they will not send an INTRESTED message to it. Second, all peers will not send REQUEST message to the free riders. If no REQUEST message is sent to free riders, they will not send PIECE message to any peer. This will also affect on the number of sent CANCEL and NOT_INTRESTED messages coming from peers who just completed the download of any piece. Figure 13 explains the consequences of not sending a HAVE message from free riders.

Assuming that ‘peer A’ is a free rider; it will shake hands with ‘peer B’ which is a seeder and ‘peer C’ which is a leecher. ‘peer A’ will exchange BITFIELD messages with ‘peer B’ and ‘peer C’ to initiate the connection and exchange information about their file status. Assuming that ‘peer A’ is interested in pieces from ‘peer B’, it will send it an INTERESTED message. When it is time for ‘peer A’ to be unchoked by peer B’, it will receive an UNCHOKED message. This will be followed by a set of REQUEST and PIECE messages until the download window is closed by a CHOKE message. Assuming that ‘peer A’ completed the download of a specific piece, it will not send a HAVE message to its neighbors, including ‘peer C’. This way, ‘peer C’ will not exchange more messages with ‘peer A’ as it has no record indicating that ‘peer A’ completed the download of any piece.

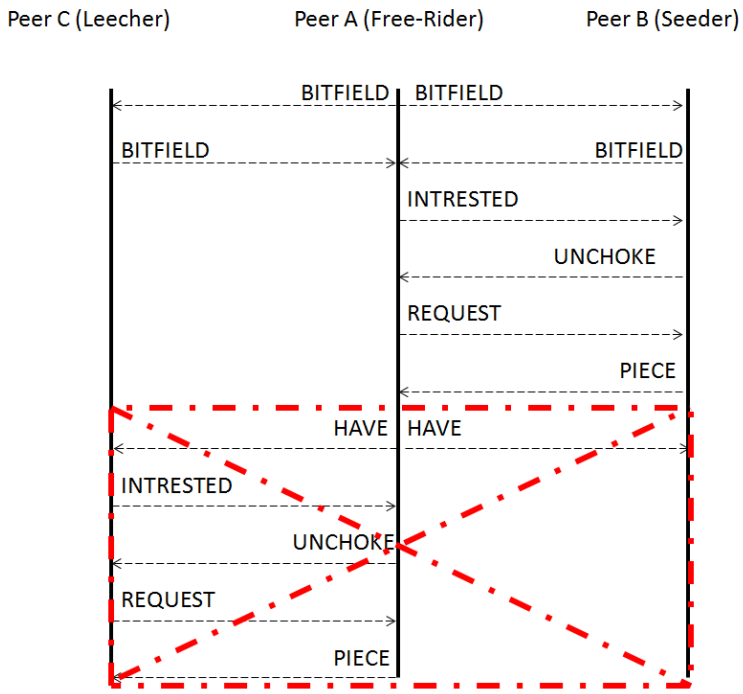


**Figure 13: The impact of not sending a HAVE message in the messages sequence**

## 5.3 The Effect of Bandwidth in the Simulation

In the simulation, there are four different bandwidths that are randomly distributed among the peers. Free riders and non-free riders could be assigned any of the four possible bandwidth values. The four available bandwidths are 640,1024, 2048 and 4096 KBps.

Before introducing free riders, figure 13 shows that peers with the lowest bandwidth (640 KBps) complete the file at the end of the simulation. The peers are ordered by file download time.

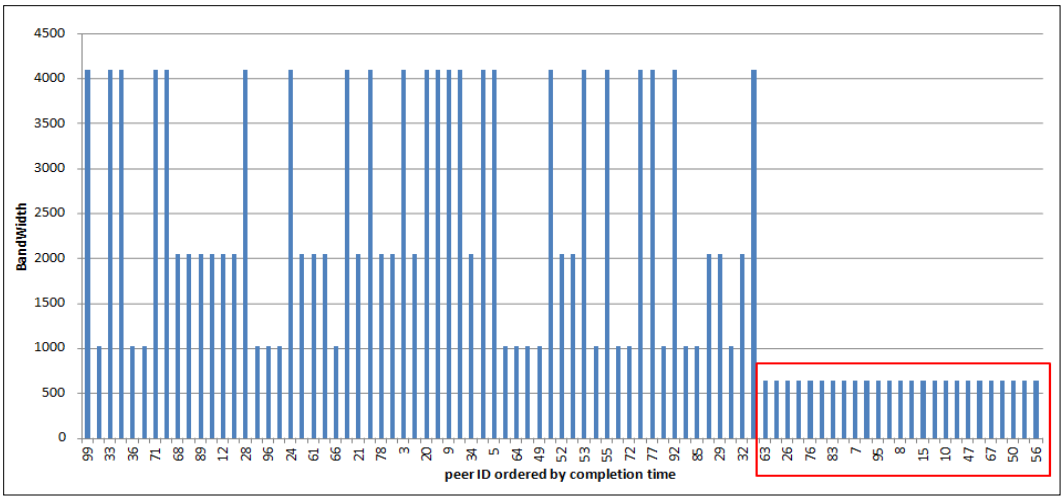


**Figure 14: Peers bandwidth and completion time without free riders**

After injecting 40% of free riders in the environment, the simulation shows that some free riders with the lowest bandwidth can get the file faster than non-free riders with higher bandwidths. Free riding peers are marked (FR) in figure 15 and they have a bandwidth of 640 KBps while a non-free rider (NFR) with higher bandwidth completed the file much later than free riders.

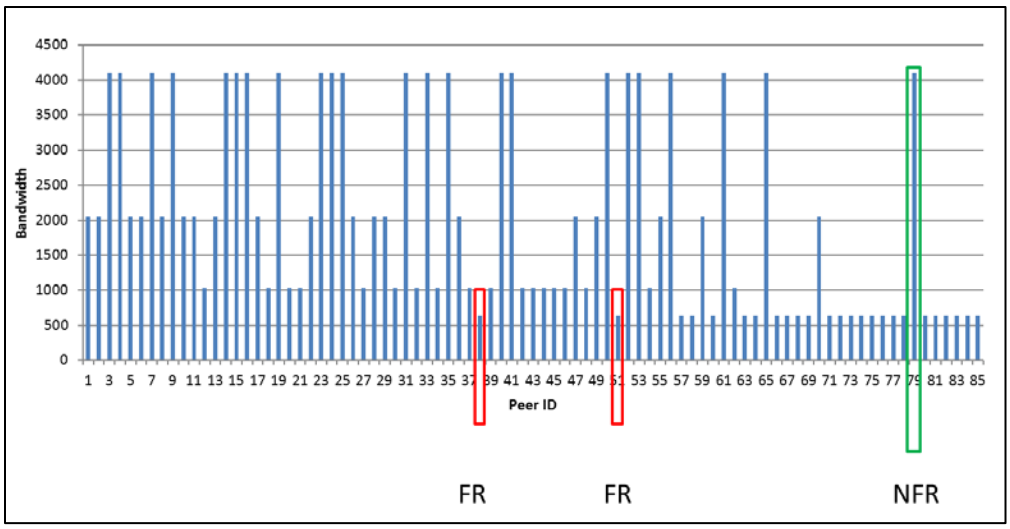


**Figure 15: Peers bandwidth and completion time with free riders**

## 5.4 Unfairness of BitTorrent Choking Algorithm

Our experiments proved that free riders can download files almost as efficient as non-free riders, sometimes even better. This means that the original choking algorithm is not designed to detect or at least delay the progress of free riders downloading a specific file. The fairness of the system is measured by the rate on which leechers are transferred into seeders, this include free riders and non-free riders.

All experiments in this section were repeated 5 times with different random seed for each experiment. The average transformation rate for leechers to become seeders is computed for free riders and non-free riders and demonstrated with different percentages of free riders and initial seeders in the environment.

The first set of experiments was generated for a network with a size of 100 peers, 15% of peers are initial seeders, these are peers who started the simulation with a complete file and their objective is only to distribute the file. The percentage of free riders configured for each set of experiments is 10%, 40% and 70%.

Figure 16, 17 and 18 demonstrate the results of leechers to seeders conversion rate for an environment with 15% of initial seeders as a fixed parameter and the percentage of free riders are 10%, 40% and 70%. A snapshot of the system is taken at equal intervals separated by 100,000 ms and this is the X axis.

Figure 16 represents the transformation rate from leechers to seeders for 40% free riders and, this percentage is selected to make an equal population of free riders and non-free

riders. The chart shows that the free riders are getting the file in a rate that is slightly faster than non-free riders. This is surly not fair for non-free riders and does not encourage them to share.

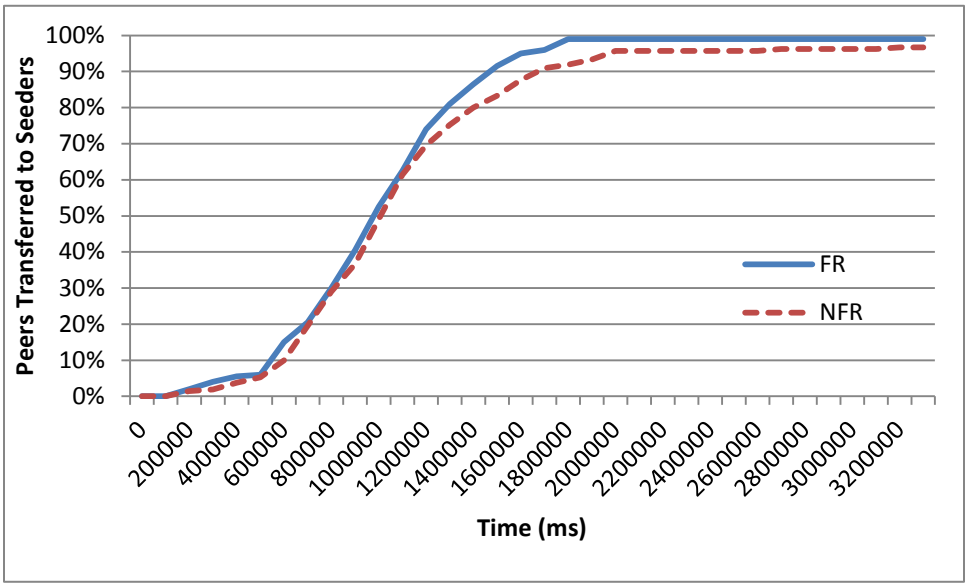


**Figure 16: Unfairness of BitTorrent choking algorithm for equal population of free riders and non-free riders**

| # of Peers | 100 | % of Free riders | 40 | % of initial seeders | 15 |
|---|---|---|---|---|---|

The same experiment was also run for 10% free riders and 70% free riders. The results are demonstrated in the figure 17 and 18:

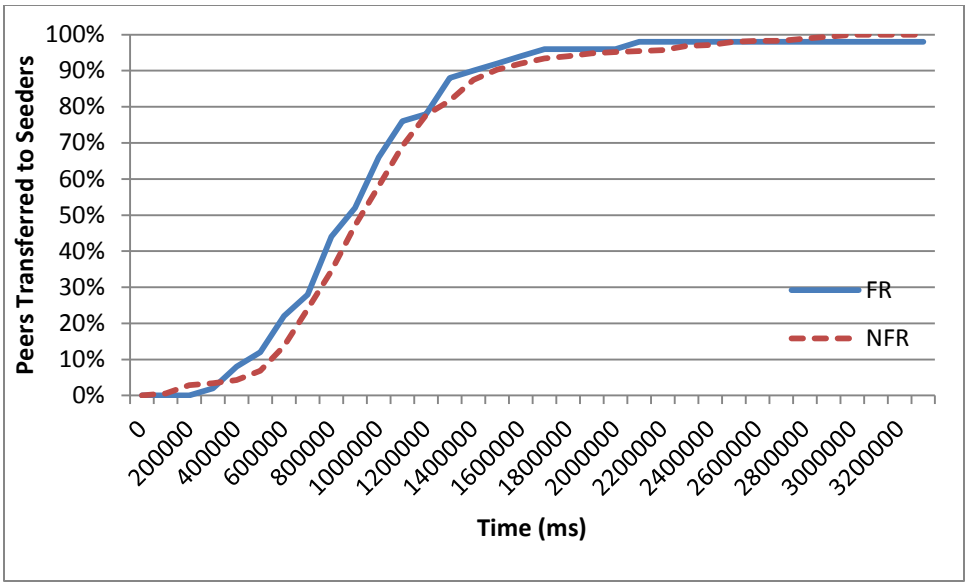


**Figure 17: Unfairness of BitTorrent choking algorithm for 10% of free riders and 15% initial seeders**

| # of Peers | 100 | % of Free riders | 10 | % of initial seeders | 15 |
|---|---|---|---|---|---|

Figure 17 shows that even for 10% free riders, they were able to get the complete file before the non-free riders and that is around time 2800000 ms where the 10 free riders downloaded the file completely. Figure 18 shows the same experimental results for 70% of free riders. The chart shows that free riders and non-free riders were able to get the complete file at the same time (2200000 ms) although that 70% of the peers are free riders. This proves that free riders generally download the file in a faster rate when compared with non-free riders.

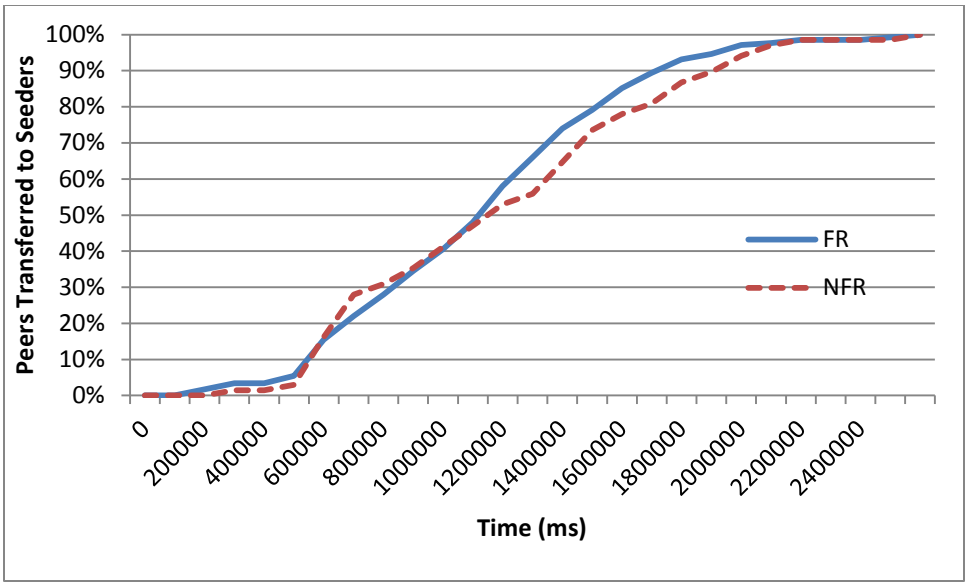


**Figure 18: Unfairness of BitTorrent choking algorithm for 70% of free riders 15% initial seeders**

| # of Peers | 100 | % of Free riders | 70 | % of initial seeders | 15 |
|---|---|---|---|---|---|

Another set of experiments were also run for different percentages of initial seeders. The objective is to analyze unfairness of BitTorrent choking algorithm even with lower percentages of initial seeder. There will be a separate section to discuss the effect of reducing the percentage of initial seeders, which are peers that have the complete file since the beginning of the simulation experiment. Figure 19, 20 and 21 show the rate of transformation of free riders and non-free riders to seeders with 8% initial seeders. The charts demonstrate the results for 10%, 40% and 70% of free riders.

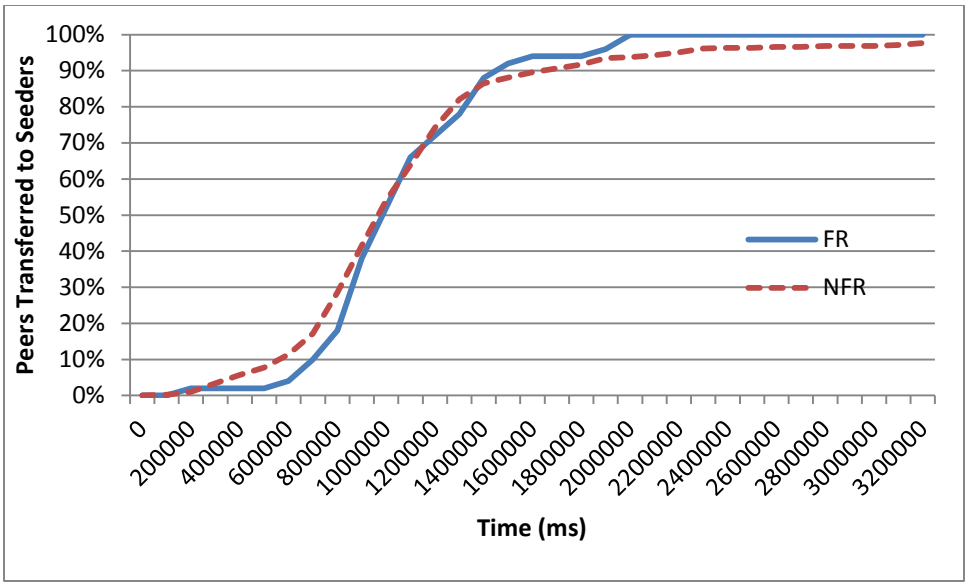


**Figure 19: Unfairness of BitTorrent choking algorithm for 10% of free riders 8% initial seeders**

| # of Peers | 100 | % of Free riders | 10 | % of initial seeders | 8 |
|---|---|---|---|---|---|

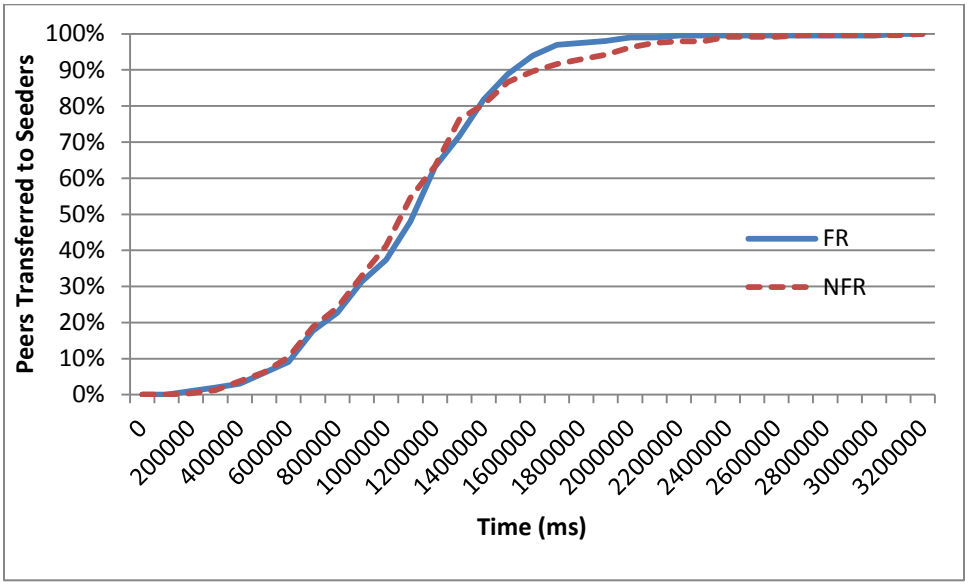


**Figure 20: Unfairness of BitTorrent choking algorithm for 40% of free riders 8% initial seeders**

| # of Peers | 100 | % of Free riders | 40 | % of initial seeders | 8 |
|---|---|---|---|---|---|

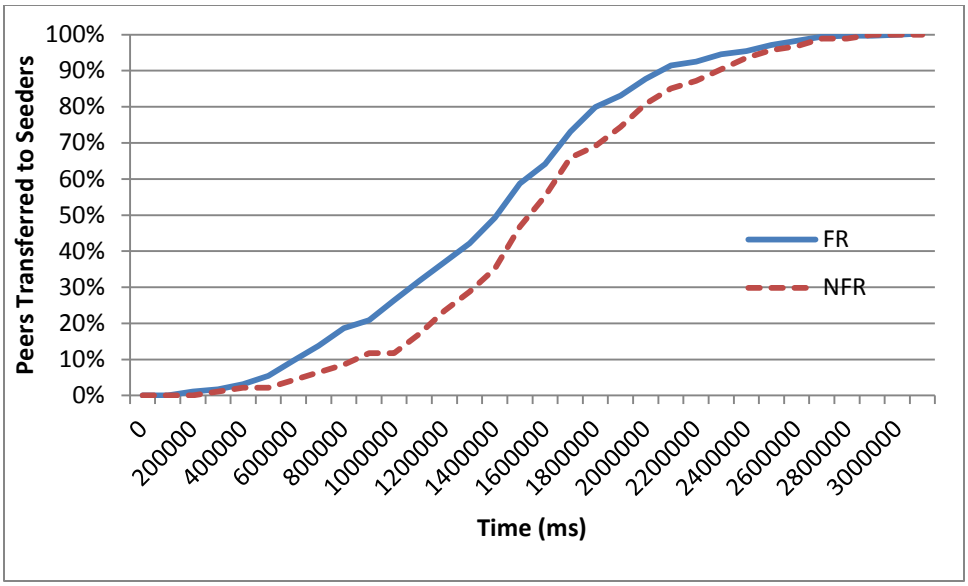


**Figure 21: Unfairness of BitTorrent choking algorithm for 70% of free riders 8% initial seeders**

| # of Peers | 100 | % of Free riders | 70 | % of initial seeders | 8 |
|---|---|---|---|---|---|

Again, figure 19, 20 and 21 still show the unfairness of the BitTorrent choking algorithm even with 8% initial seeders in the environment. It is clearly noticed that free riders are able to get the files without contributing any block to the environment.

The same set of experiments was run for 4% initial seeder percentage, also covering 10%, 40% and 70% free riders. The results in the below three charts also show a competition between the free riders and non-free riders except on the last chart where the percentage of free riders is 70%. The results in that chart shows that free riders are transforming into non-free riders with a slower rate comparing with non-free riders.

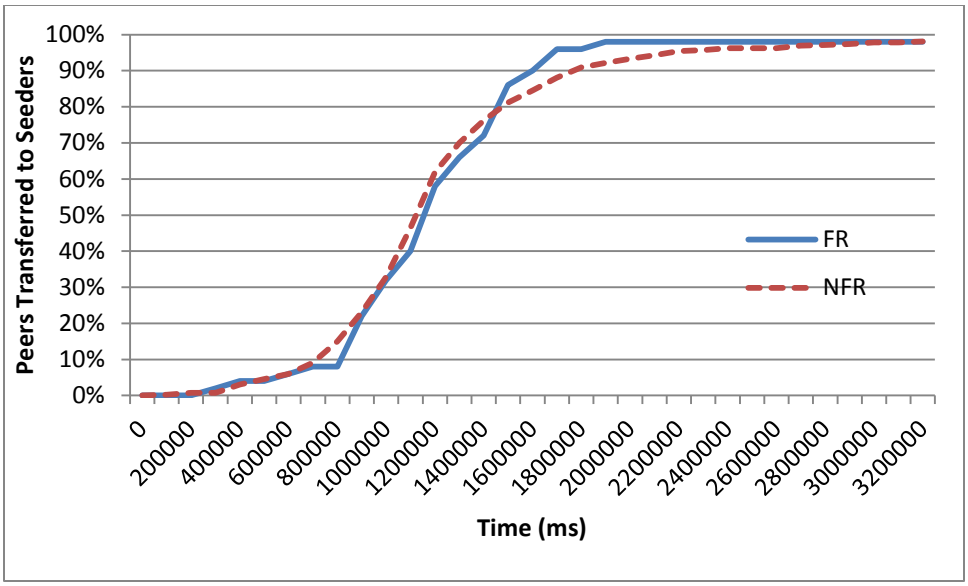


**Figure 22: Unfairness of BitTorrent choking algorithm for 10% of free riders 4% initial seeders**

| # of Peers | 100 | % of Free riders | 10 | % of initial seeders | 4 |
|---|---|---|---|---|---|

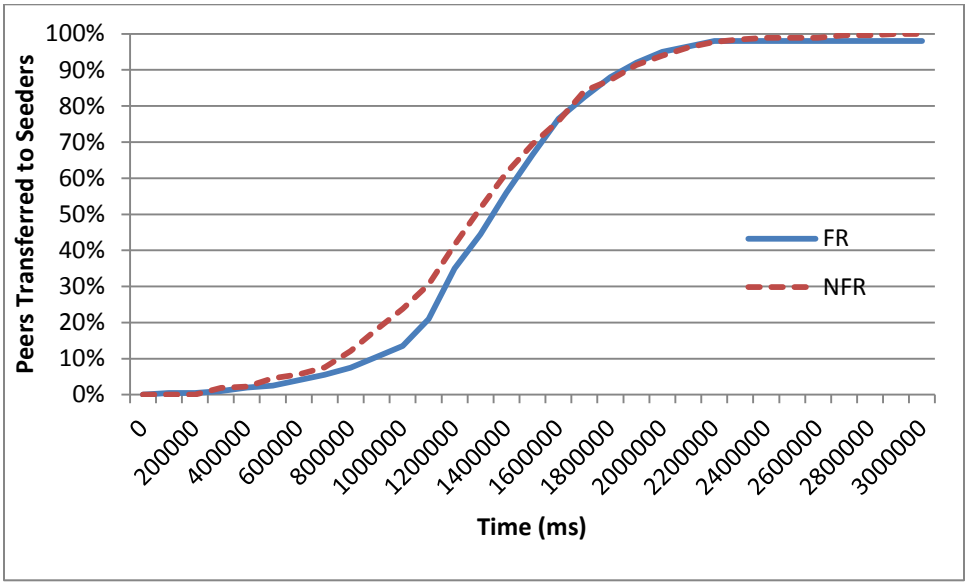


**Figure 23: Unfairness of BitTorrent choking algorithm for 40% of free riders 4% initial seeders**

| # of Peers | 100 | % of Free riders | 40 | % of initial seeders | 4 |
|---|---|---|---|---|---|

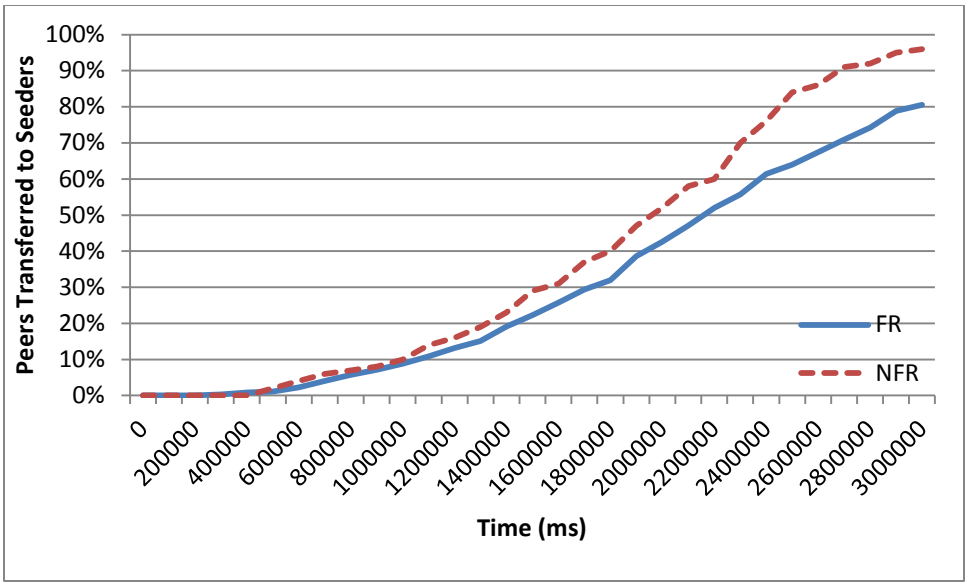


**Figure 24: Unfairness of BitTorrent choking algorithm for 70% of free riders 4% initial seeders**

| # of Peers | 100 | % of Free riders | 70 | % of initial seeders | 4 |
|---|---|---|---|---|---|

## 5.5 Effect of the Number of Initial Seeders in the Environment

This section of the thesis experimentally discusses the effect of reducing the number of the initial seeders in the environment. Seeders have no mean to detect free riders behavior. They simply unchoke three peers whether they are interested or randomly picked in every unchoking cycle. This section will cover the discussion of six figures as the following:

1. Effect of changing the percentage of initial seeders from 15% to 8% and 4% on a population of 10% free riders out of 100 peers.
2. Effect of changing the percentage of initial seeders from 15% to 8% and 4% on a population of non-free riders in an environment with 10% free riders.

3. Effect of changing the percentage of initial seeders from 15% to 8% and 4% on a population of 40% free riders out of 100 peers.
4. Effect of changing the percentage of initial seeders from 15% to 8% and 4% on a population of non-free riders in an environment with 40% free riders.
5. Effect of changing the percentage of initial seeders from 15% to 8% and 4% on a population of 70% free riders out of 100 peers.
6. Effect of changing the percentage of initial seeders from 15% to 8% and 4% on a population of non-free riders in an environment with 70% free riders.

Figure 25 shows that for 10% free riders, in the experiments which were run for 4% initial seeders (solid line); free riders were slower in transforming into seeders. The chart also shows that on the experiments that were run for 8% initial seeders (dotted line), free riders were slower in transforming to seeders when compared with the free riders in an environment with 15% initial seeders (dashed line). This shows a direct relationship between the percentage of initial seeders and the rate on which free riders are converted to seeders when their population in the environment is 10%.

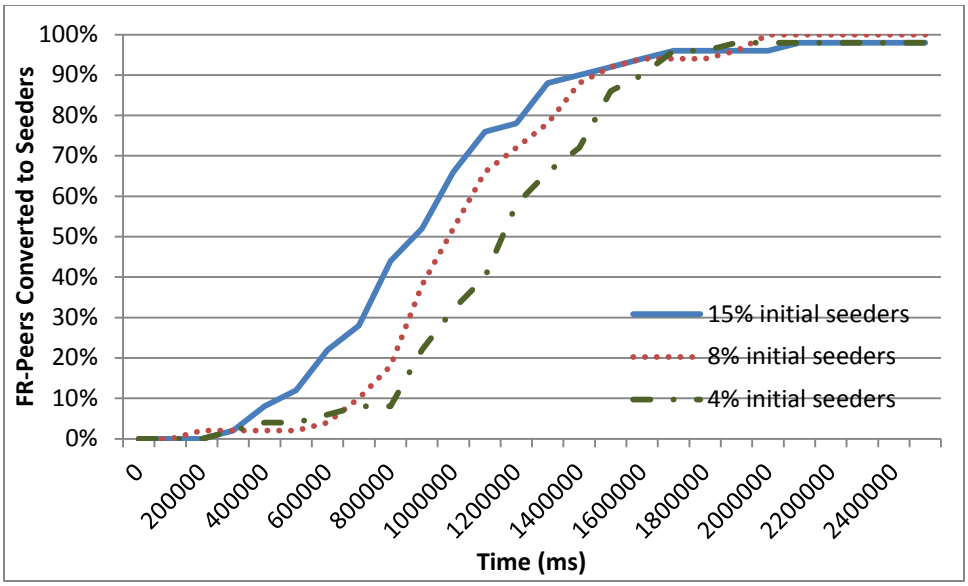


**Figure 25: The effect of changing the initial seeders % on the download progress of 10% free riders**

| # of Peers | 100 | % of Free riders | 10 | % of initial seeders | 15,8,4 |
|---|---|---|---|---|---|

For the remaining non-free riders, which are not considered as initial seeders after labeling 10% free riders, the effect of initial seeders reduction between 15% and 8% does not have a large impact on the transformation rate into seeders. However, the impact is greater when the percentage of free riders is reduced to 4%. This shows that for an environment with 10% free riders, the effect of reducing the number of initial seeders has a higher impact on the download progress of free riders than the impact on non-free riders. Figure 26 shows this relationship. It also apparent that non-free riders were able to catch-up at the end of the experiment (1800000 ms) and this is due to having different numbers of non-free riders in the environment as this is related to the distribution of the initial seeders in the environment.

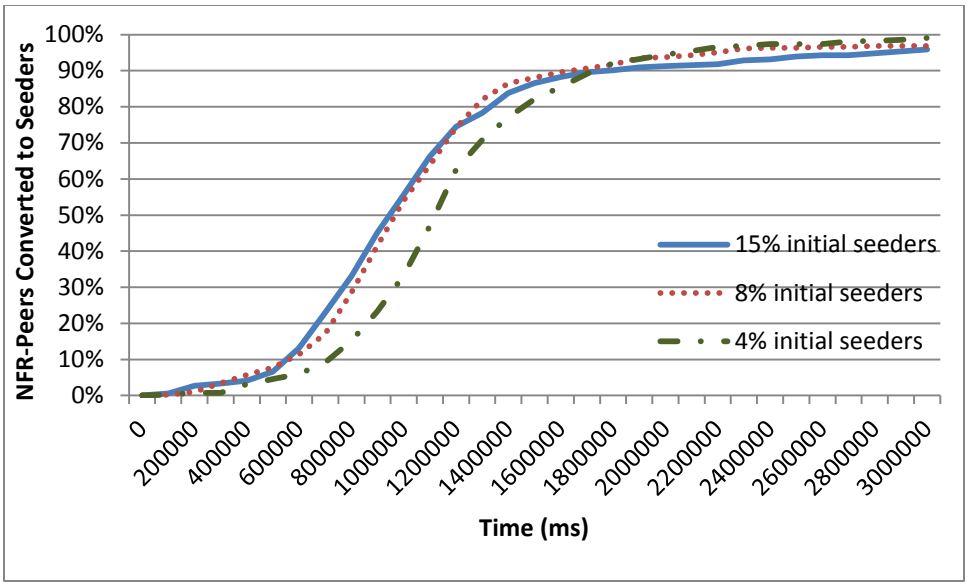


**Figure 26: The effect of changing the initial seeders % on the download progress of non-free riders with 10% free riders**

| # of Peers | 100 | % of Free riders | 10 | % of initial seeders | 15,8,4 |
|---|---|---|---|---|---|

The same conclusion is also applicable for an environment with 40% free riders. Figure 27 shows that free riders transformation into seeders is highly related to the number of initial seeders. Again, seeders have no mean to detect free riding behavior and thus there number in the environment highly affects the download progress of free riders. For non-free riders, the impact on their download progress is highly visible when the percentage of initial free riders is 4%, while the difference in download progress is minimal between an environment with 15% initial seeders and an environment with 8% free riders. Figure 27 and 28 are used to illustrate this conclusion.

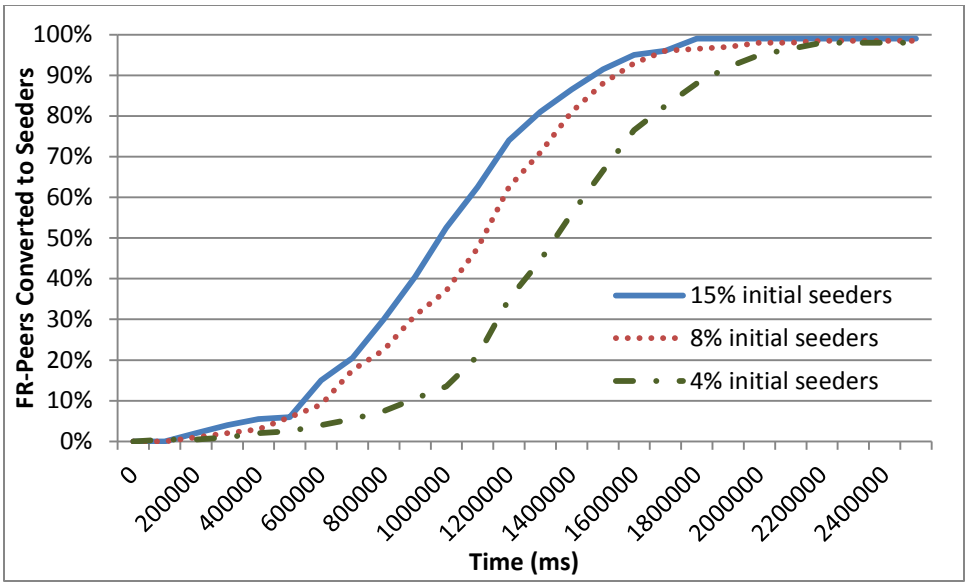


**Figure 27: The effect of changing the initial seeders % on the download progress of 40% free riders**

| # of Peers | 100 | % of Free riders | 40 | % of initial seeders | 15,8,4 |
|---|---|---|---|---|---|

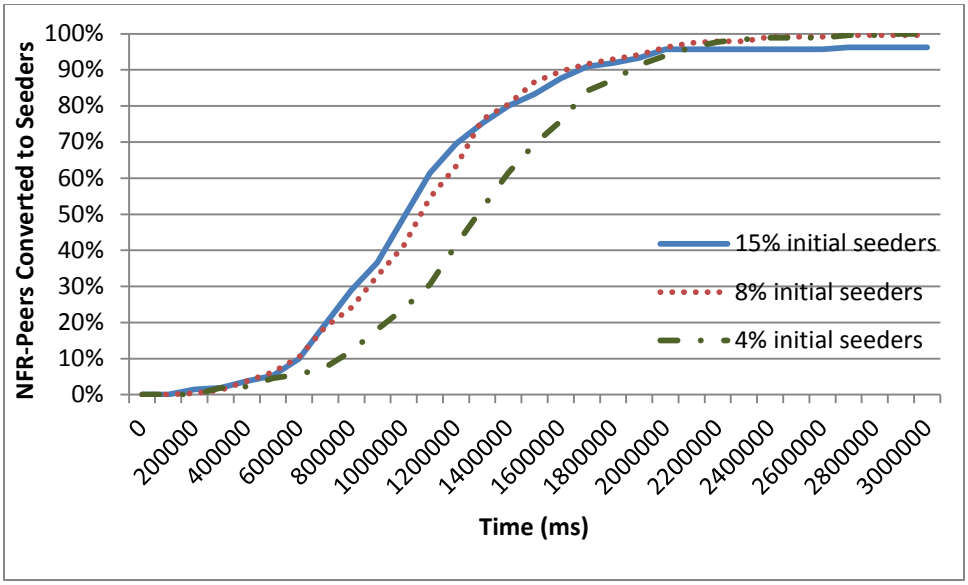


**Figure 28: The effect of changing the initial seeders % on the download progress of non-free riders with 40% free riders**

| # of Peers | 100 | % of Free riders | 40 | % of initial seeders | 15,8,4 |
|---|---|---|---|---|---|

The same experiment was also run for an environment with 70% free riders. This time, the demand on the initial seeders is more as the population of free riders is high and thus the resources are lower in the environment. In this case, both free riders and non-free riders download progress is highly affected by changing the percentage of initial seeders in the environment. It is clearly visible that for an environment with 4% initial seeders (scattered line) where the time needed for free riders to complete the download the file is increased to a large extent. Non-free riders download performance is also decreased as the number of initial seeders decreases. These two conclusions are illustrated in figure 29 and 30.

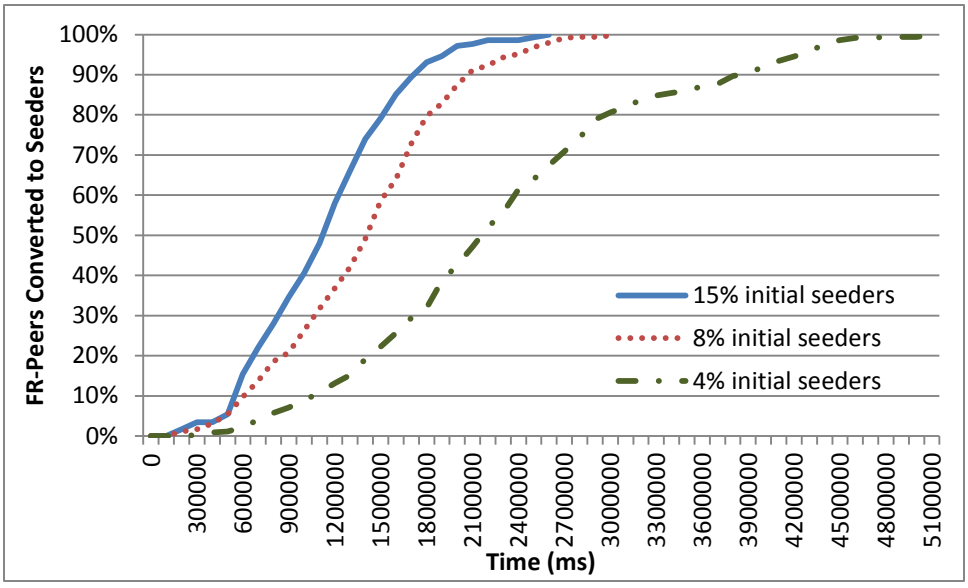


**Figure 29: The effect of changing the initial seeders % on the download progress of 70% free riders**

| # of Peers | 100 | % of Free riders | 70 | % of initial seeders | 15,8,4 |
|---|---|---|---|---|---|

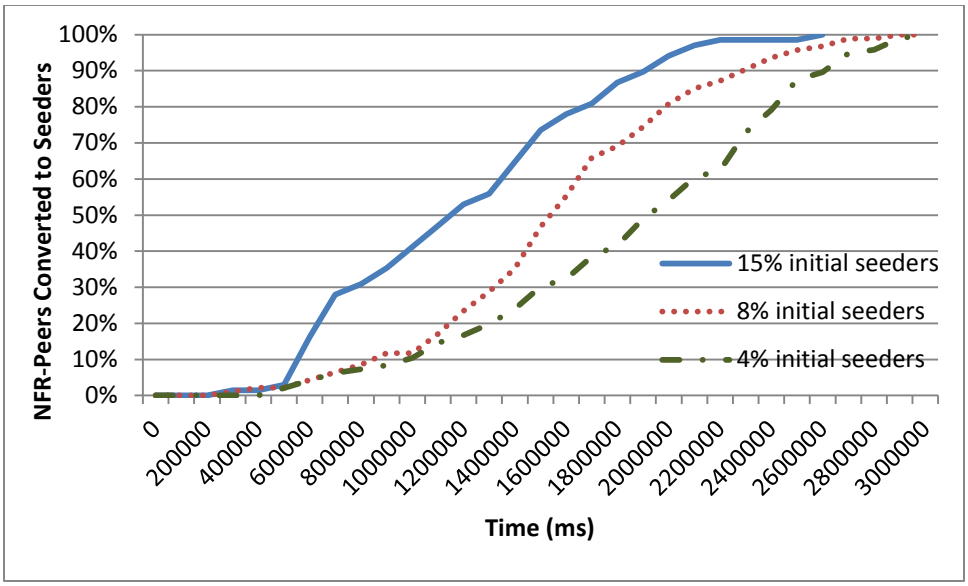


**Figure 30: The effect of changing the initial seeders % on the download progress of non-free riders with 70% free riders**

| # of Peers | 100 | % of Free riders | 70 | % of initial seeders | 15,8,4 |
|---|---|---|---|---|---|

## 5.6 Measuring the Improvement of Our Approach

This section of the thesis demonstrates the effect of our modified choking algorithm in slowing down the download progress of free riders in the system. The table below each chart shows the parameters that are used to capture the results. Figures 31 and 32 are run with 10% free riders in the system, 100 peers network size, 15 % initial seeders distribution for an average of 5 experiments. Figure 31 shows the transformation rate of free riders from leechers to seeders and figure 32 shows the transformation rate of non-free riders from leechers to seeders.

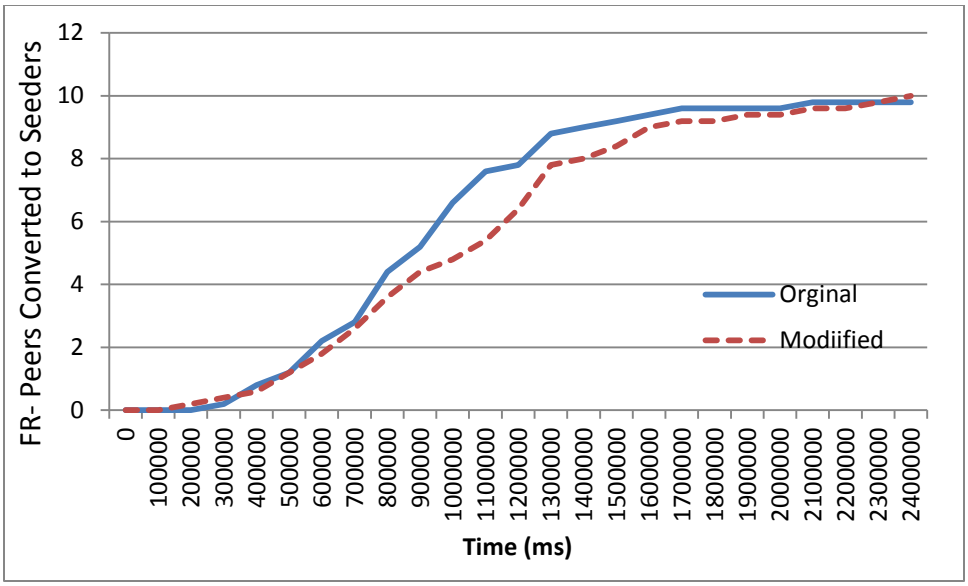


**Figure 31: Transformation of free riders into Seeders (10% free riders, 15% initial seeders)**

| # of Peers | 100 | % of Free riders | 10 | % of initial seeders | 15 |
|---|---|---|---|---|---|

Figure 31 shows the improvement that is gained by applying the X chance giving algorithm. Scattered line shows the transformation rate of free riders from leechers to seeders after improvement, while solid line shows the transformation rate of free riders from leechers to seeders with the original choking algorithm. The experiment shows that the effect of the modified algorithm needs time until it gives a noticeable effect. This is due to the fact that all peers that apply the algorithm will give X-chances until they start punishing free riders. The punishment here is achieved by reducing the number of UNCHOKE messages that free riders receive from leechers and seeders with experience. The figure also shows that at the end of the experiment, the original choking algorithm and the modified choking algorithm transformation rate will be close to each other. This is due to the fact that initial seeders are still providing free riders with pieces and by this time all non-free riders already completed the download of the file.

Figure 32 shows the transformation rate from seeders to leechers for non-free riders.

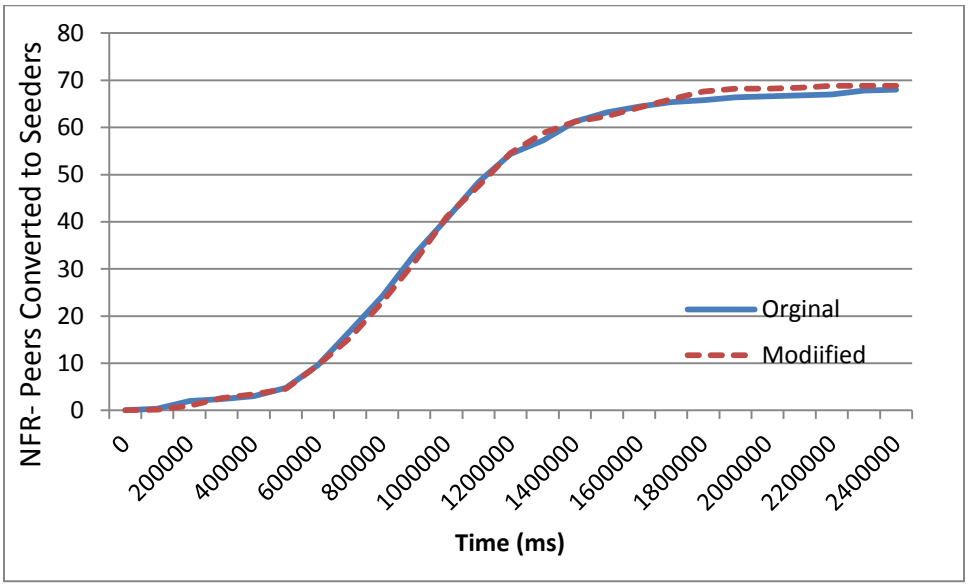


**Figure 32: Transformation of non-free riders into seeders (10% free riders, 15% initial seeders)**

| # of Peers | 100 | % of Free riders | 10 | % of initial seeders | 15 |
|---|---|---|---|---|---|

Figure 32 shows that the algorithm was able to maintain the download progress for non-free riders and exceeded it minimally towards the end of the simulation. The reason behind maintaining the download progress for non-free riders is due to not sending UNCHOKE message to non-free rider when a free rider is detected. This is the case in the regular choking algorithm that happens every 10 seconds. However, the algorithm will send an UNCHOKE message to non-free rider when a free rider is detected in the optimistic unchoking routine that happens every 30 seconds.

The next four figures (33, 34, 35 and 36) were run for the same parameters except changing the percentage of free riders in the system. Figure 33 and 34 show the same experimental setup discussed above but with 40% and 70% free riders and this is to test the effectiveness of the modified algorithms for a larger number of free riders.

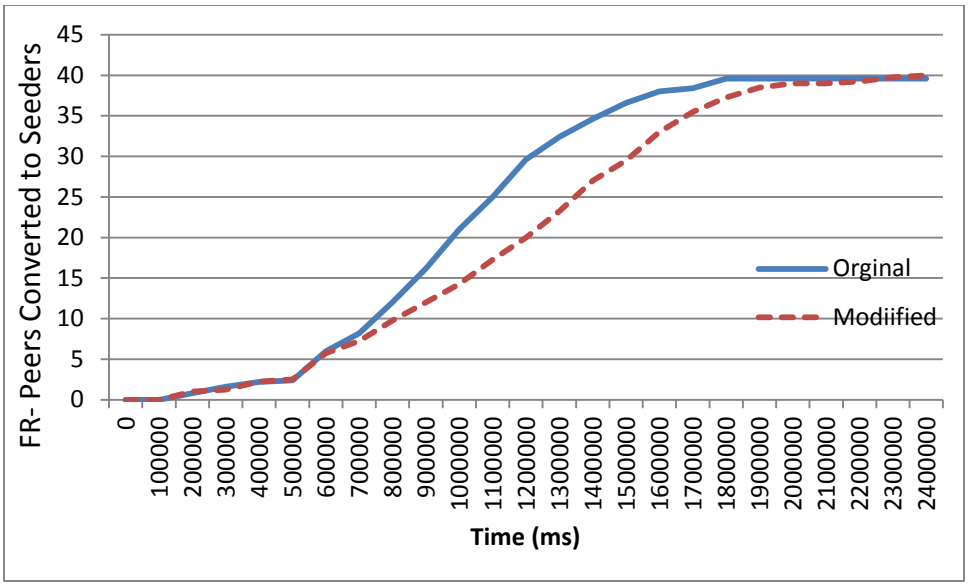


**Figure 33: Transformation of free riders into seeders (40% free riders, 15% initial seeders)**

| # of Peers | 100 | % of Free riders | 40 | % of initial seeders | 15 |
|---|---|---|---|---|---|

Figure 33 shows that the free riders transformation rate from leechers to seeders for the modified algorithm is lower than the original choking algorithm, even with 40% free riders in the environment. The same figure also shows that the algorithm 650000 ms to start being effective in slowing down the progress of free riders file downloading progress. Figure 34 shows that the modified algorithm does not have any considerable negative impact on the download performance of non-free riders. 40% free riders was selected to make an equal number of free riders and non-free riders that are not initially seeders in the system and test the algorithm under these conditions.

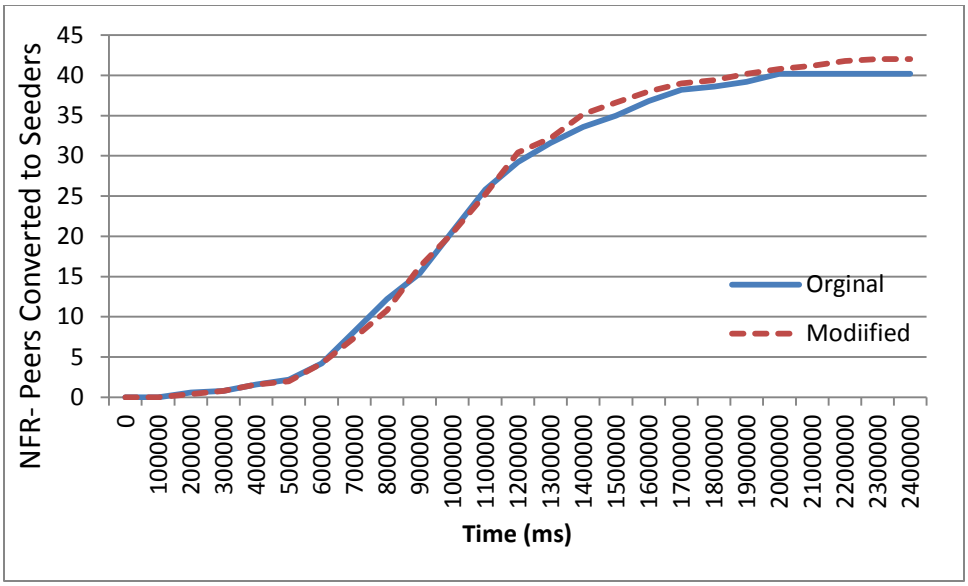


**Figure 34: Transformation of non-free riders into seeders (40% free riders, 15% initial seeders)**

| # of Peers | 100 | % of Free riders | 40 | % of initial seeders | 15 |
|---|---|---|---|---|---|

The next two figures (35 and 36) show that the modified algorithm is still effective for 70% free riders in the environment and for the same experimental setup.

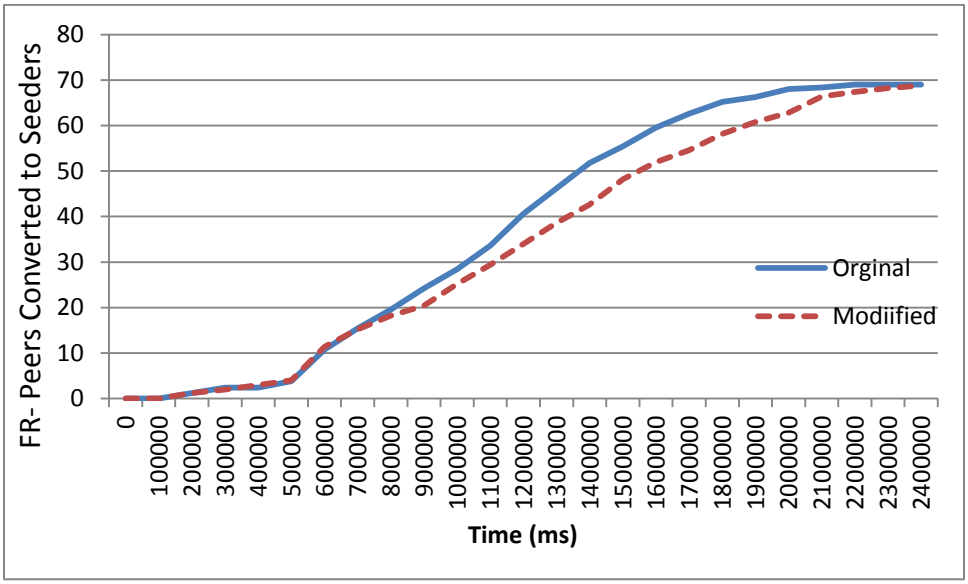


**Figure 35: Transformation of free riders into Seeders (70% free riders, 15% initial seeders)**

| # of Peers | 100 | % of Free riders | 70 | % of initial seeders | 15 |
|---|---|---|---|---|---|

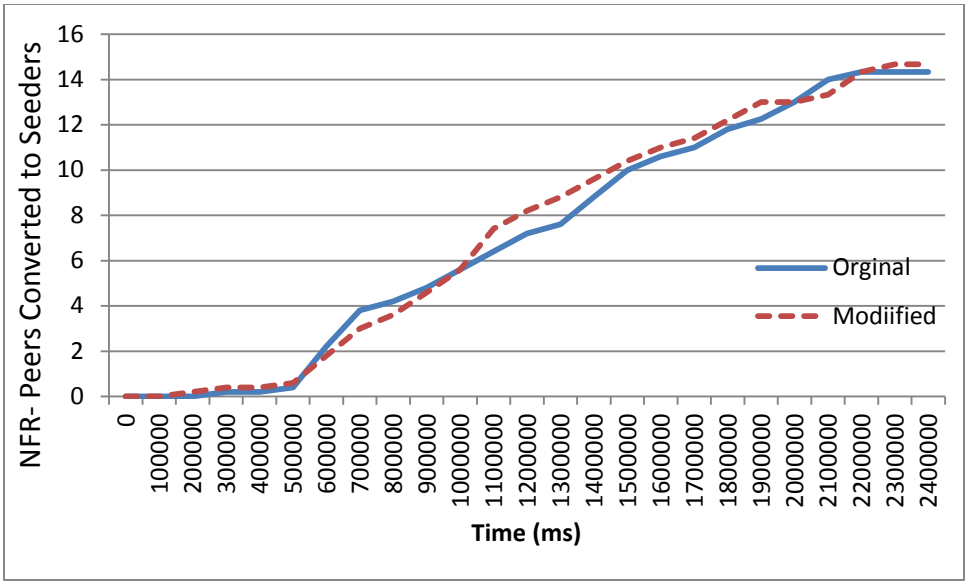


**Figure 36: Transformation of non-free riders into seeders (70% free riders, 15% initial seeders)**

| # of Peers | 100 | % of Free riders | 70 | % of initial seeders | 15 |
|---|---|---|---|---|---|

It is clearly apparent that the modified algorithm is able to maintain the download rate for the non-free riders and thus they were transformed to seeders in the same rate before and after modifying the choking algorithm. However, for free riders, they were slowed-down to an extent in all the results displayed above.

As mentioned in the literature and proven through our experiments, free riders mostly use the seeders to get their resources as seeders has no way to detect the selfish behavior of a free rider. As a result, the same experiment was run with a lower number of initial seeders who have no experience to relay on to detect free riders behavior. Below two charts (37 and 38) show the results for 100 peers, 10% free riders and 8% of initial seeders.

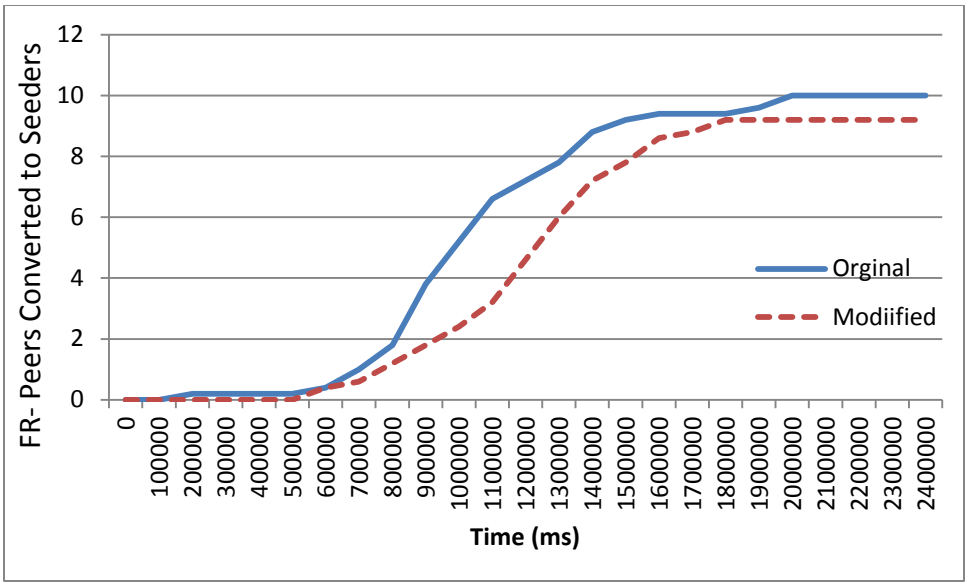


**Figure 37: Transformation of free riders into Seeders (10% free riders, 8% initial seeders)**

| # of Peers | 100 | % of Free riders | 10 | % of initial seeders | 8 |
|---|---|---|---|---|---|

For an environment with 8% initial seeders and 10% free riders, the impact of the modified algorithm increases comparing with results for an environment with 15% initial seeders. The gap between the two lines figure 37 is relatively large and it takes longer time for all free riders to get the complete file. In the above figure, only 9 free riders were able to get the file in the reported time. The remaining one peer was able to download the file but much later in simulation time. Figure 38 still shows that the download performance for non-free rider is not negatively impacted with the modified algorithm; except a slight reduction between the time 7000000 and 10000000.

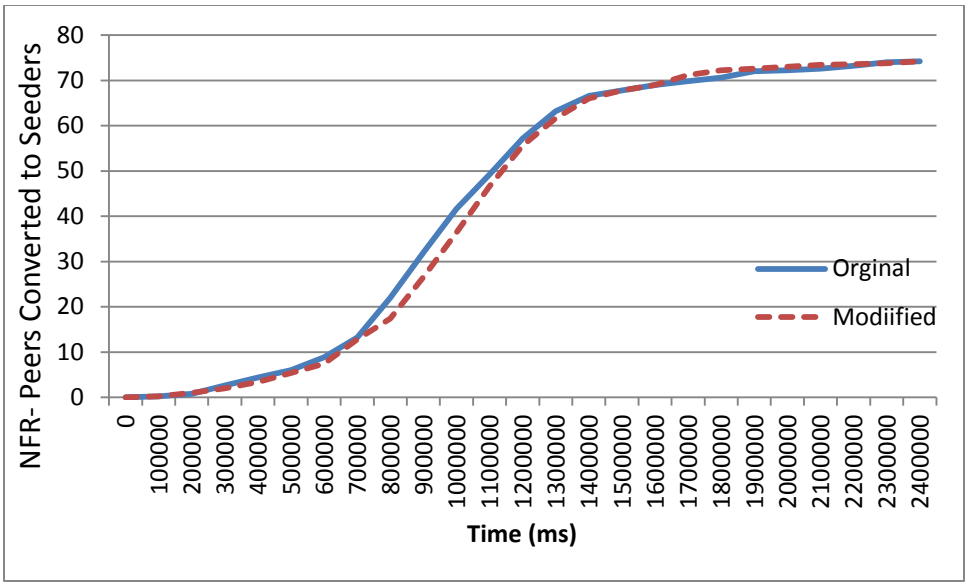


**Figure 38: Transformation of non-free riders into seeders (70% free riders, 8% initial seeders)**

| # of Peers | 100 | % of Free riders | 10 | % of initial seeders | 8 |
|---|---|---|---|---|---|

The same experimental setup was also run for 40% and 70 free riders. The results for free riders and non-free rider transformation rate from leechers to seeders are displayed in the figures 39, 40, 41 and 42.

The results show that the algorithm is more effective in an environment with less percentage of initial seeders. Also the algorithm is more effective with smaller number of free riders as the level of punishment in the environment is greater. Figure 40 shows a noticeable improvement in an environment with 70% free riders. This is due to unchoking a non-free rider every time a free rider is detected and not being unchoked in the optimistic unchoking.

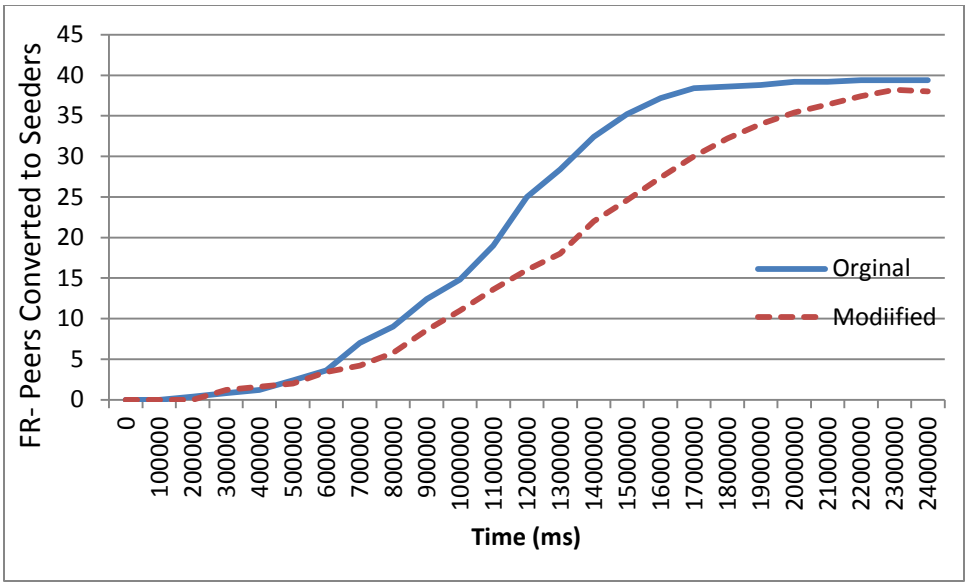


**Figure 39: Transformation of free riders into Seeders (40% free riders, 8% initial seeders)**

| # of Peers | 100 | % of Free riders | 40 | % of initial seeders | 8 |
|---|---|---|---|---|---|

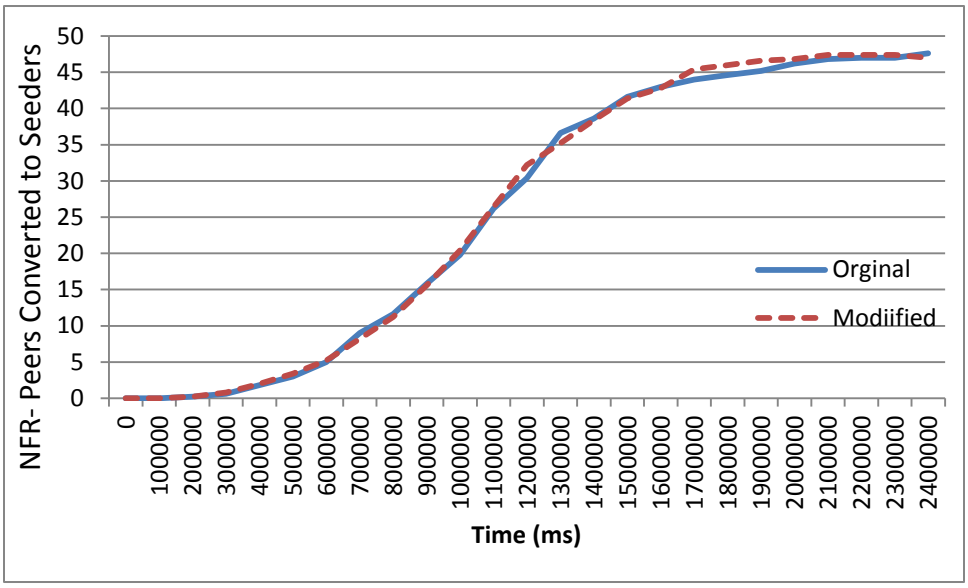


**Figure 40: Transformation of non-free riders into seeders (40% free riders, 8% initial seeders)**

| # of Peers | 100 | % of Free riders | 40 | % of initial seeders | 8 |
|---|---|---|---|---|---|

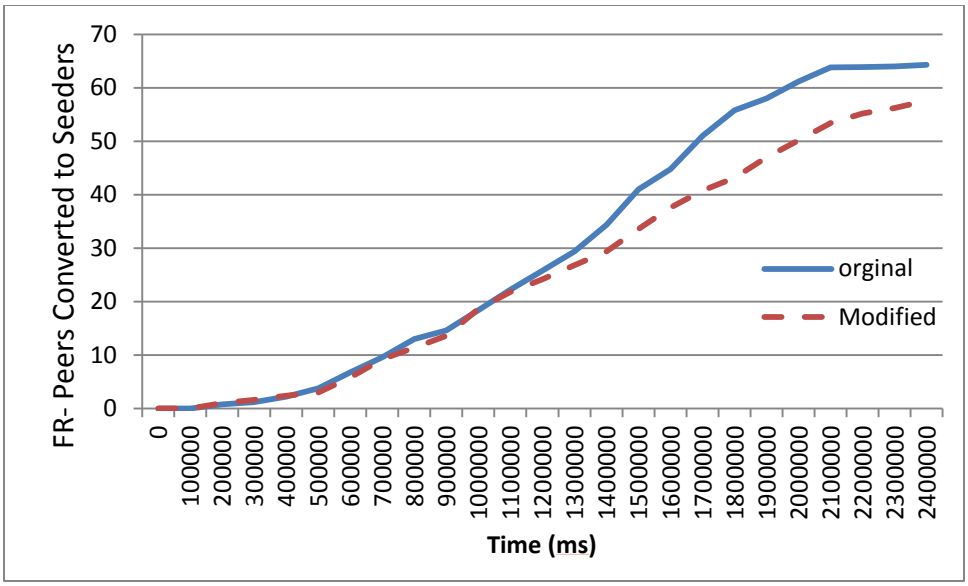


**Figure 41: Transformation of free riders into Seeders (70% free riders, 8% initial seeders)**

| # of Peers | 100 | % of Free riders | 70 | % of initial seeders | 8 |
|---|---|---|---|---|---|

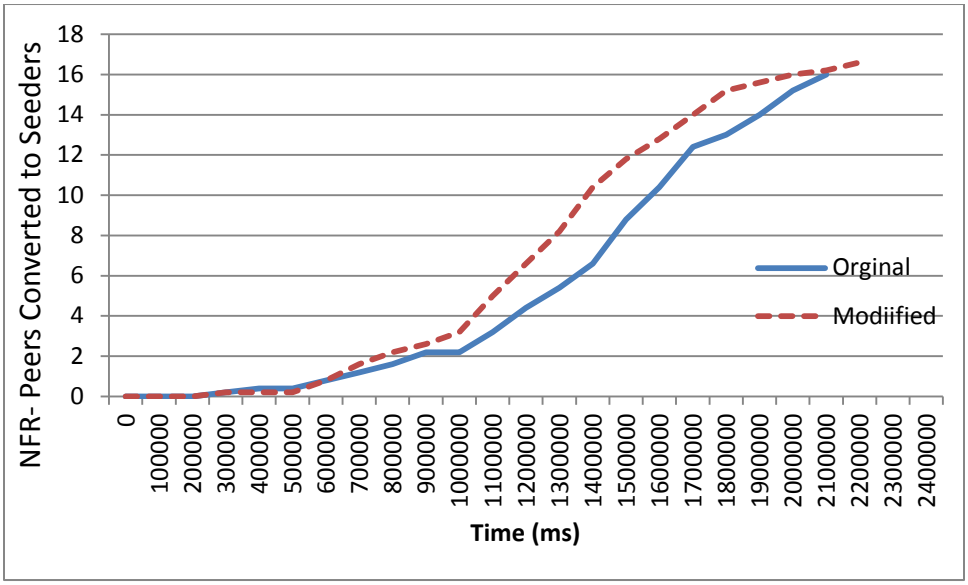


**Figure 42: Transformation of non-free riders into seeders (70% free riders, 8% initial seeders)**

| # of Peers | 100 | % of Free riders | 70 | % of initial seeders | 8 |
|---|---|---|---|---|---|

To further study the effect of reducing the number of the initial seeders in the environment, the same set of experiments were run with 4% initial seeders. The transformation rate from leechers to seeders for free riders is reduced sharply when the percentage of free riders in the environment is 10%. This is illustrated in figure 43. For this experiment, the modified algorithm starts to make an effect around 80000 ms. It is also important to mention that the modified choking algorithm was able to slowdown two peers out of ten to a long time. Figure 43 shows that the modified algorithm transformation rate was not completed at time 24000000.

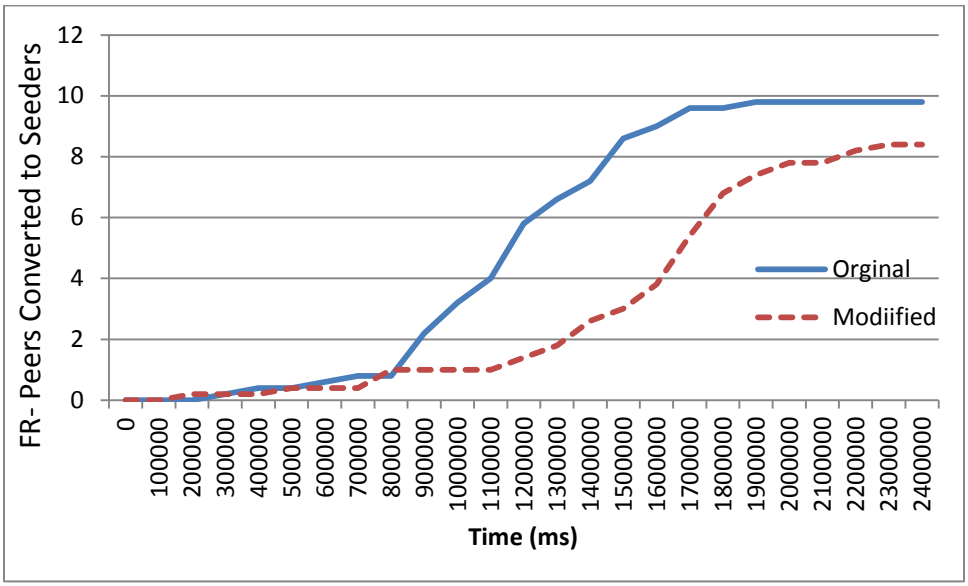


**Figure 43: Transformation of free riders into Seeders (10% free riders, 4% initial seeders)**

| # of Peers | 100 | % of Free riders | 10 | % of initial seeders | 4 |
|---|---|---|---|---|---|

As illustrated in figure 44, non-free riders are maintaining the same transformation rate before and after the modification of the choking algorithm.

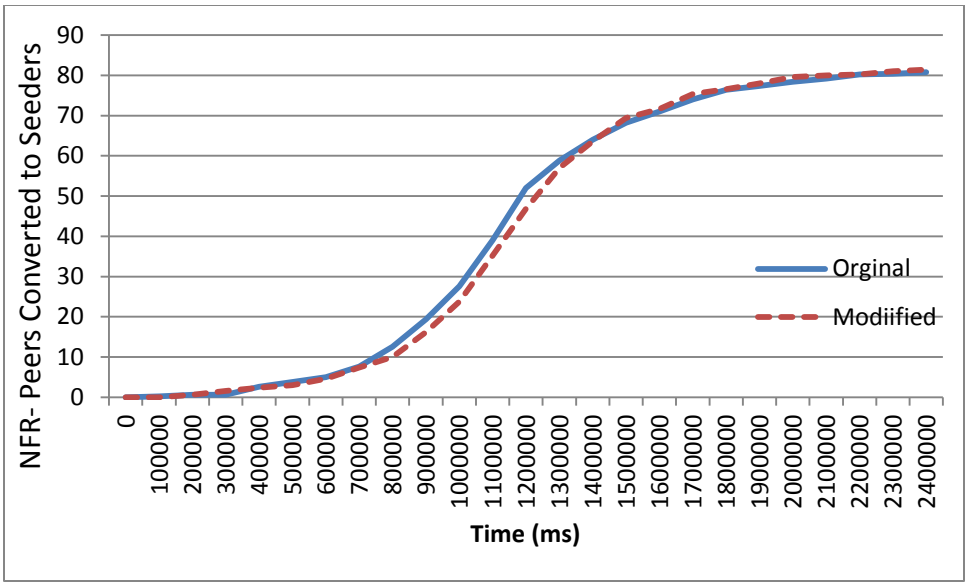


**Figure 44: Transformation of non-free riders into seeders (10% free riders, 4% initial seeders)**

| # of Peers | 100 | % of Free riders | 10 | % of initial seeders | 4 |
|---|---|---|---|---|---|

The same experimental setup was also run for 40% and 70 free riders. The results for free riders and non-free rider transformation rate from leechers to seeders are displayed in figures 45, 46, 47 and 48.

Figure 45 shows the results of applying the modified choking algorithm in an environment with 40% free riders and 4% initial seeders shows a noticeable decrease in the downloading performance of non-free riders. At the end of the experiment monitoring time, there are 10 free riders that were not able to complete the file download. The algorithm was also able to maintain the download performance for non-free riders as shown in figure 46.

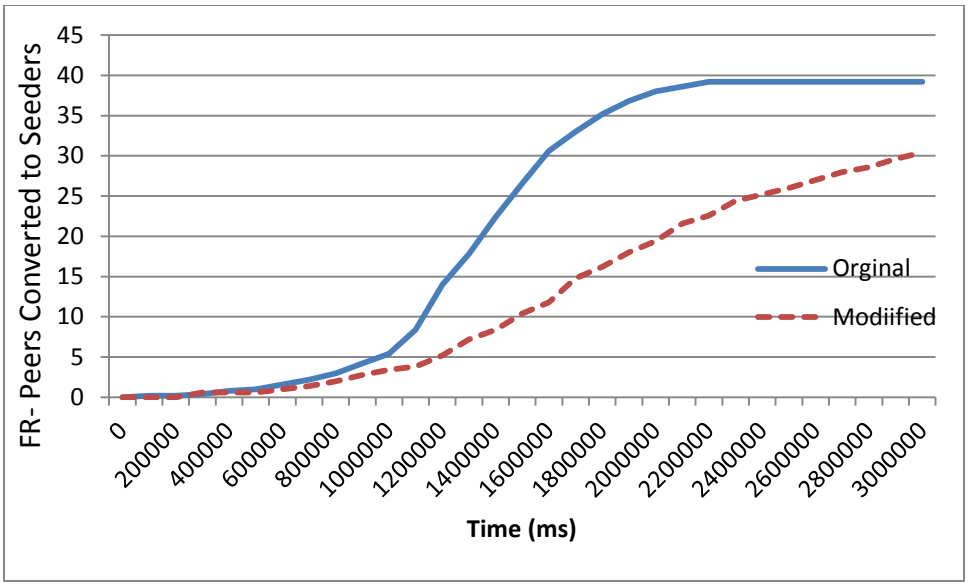


**Figure 45: Transformation of free riders into Seeders (40% free riders, 4% initial seeders)**

| # of Peers | 100 | % of Free riders | 40 | % of initial seeders | 4 |
|---|---|---|---|---|---|

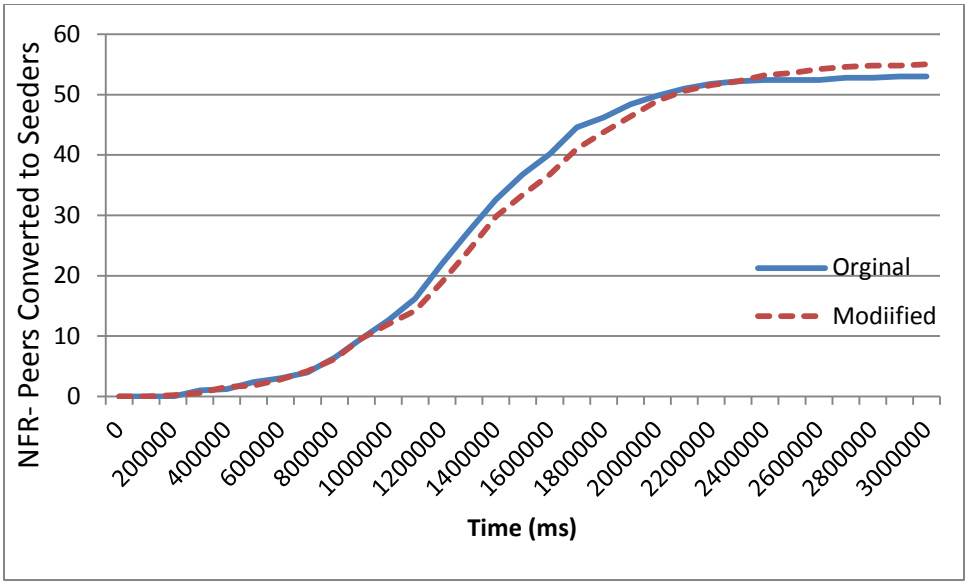


**Figure 46: Transformation of non-free riders into seeders (40% free riders, 4% initial seeders)**

| # of Peers | 100 | % of Free riders | 40 | % of initial seeders | 4 |
|---|---|---|---|---|---|

The experiment is also run for 70% free riders in the environment and 4% initial seeders. The results show that the modified algorithm is able to reduce the performance of the file download by the free riders. The experiment was recorded fully, but what is reported is only the major part of it. At 3000000 ms, around 30 peers completed the download of the file after modifying the choking algorithm. At the same time, around 55 peers completed the download of the file with the original choking algorithm.

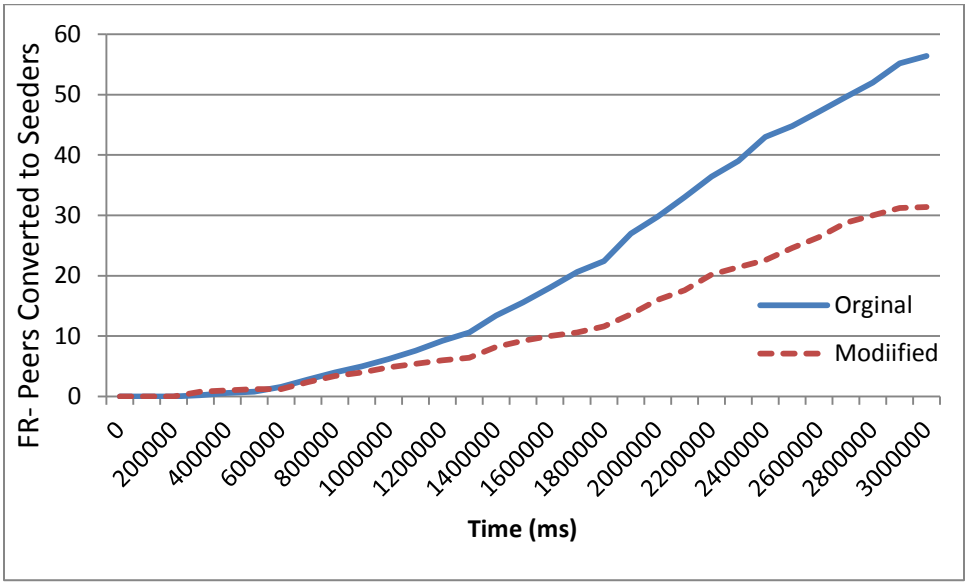


**Figure 47: Transformation of free riders into Seeders (70% free riders, 4% initial seeders)**

| # of Peers | 100 | % of Free riders | 70 | % of initial seeders | 4 |
|---|---|---|---|---|---|

Figure 48 also shows that the modified algorithm was able to maintain the download performance for non-free rider.

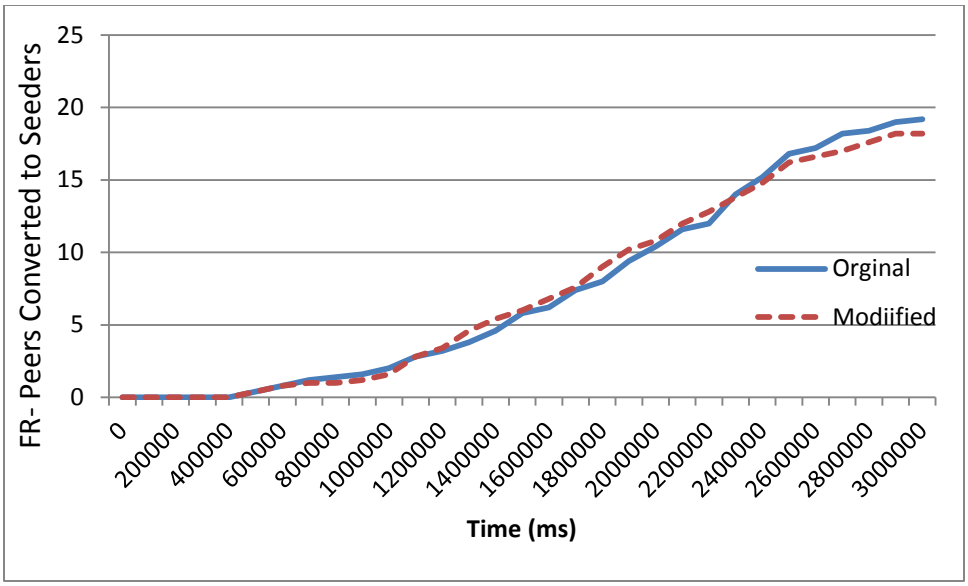


**Figure 48: Transformation of non-free riders into seeders (70% free riders, 4% initial seeders)**

| # of Peers | 100 | % of Free riders | 70 | % of initial seeders | 4 |
|---|---|---|---|---|---|

All the experiments mentioned above are run for a network with a size of 100 peers. The results show an average of five experiments for each chart. In order to verify that the modified algorithm also works with larger network sizes, one experiment with 1000 peers and 5% of initial seeders was run and the results are discussed below. The same experimental setup was run for 10%, 40% and 70% free riders.

Figure 49 shows that for 10% free riders in the environment, there is a huge improvement in delaying the free rider download performance. This can be noticed at time 1600000 ms, where around 60 free riders got the complete file. However, all free riders got the file completely when the original choking algorithm is utilized at time 1600000 ms.

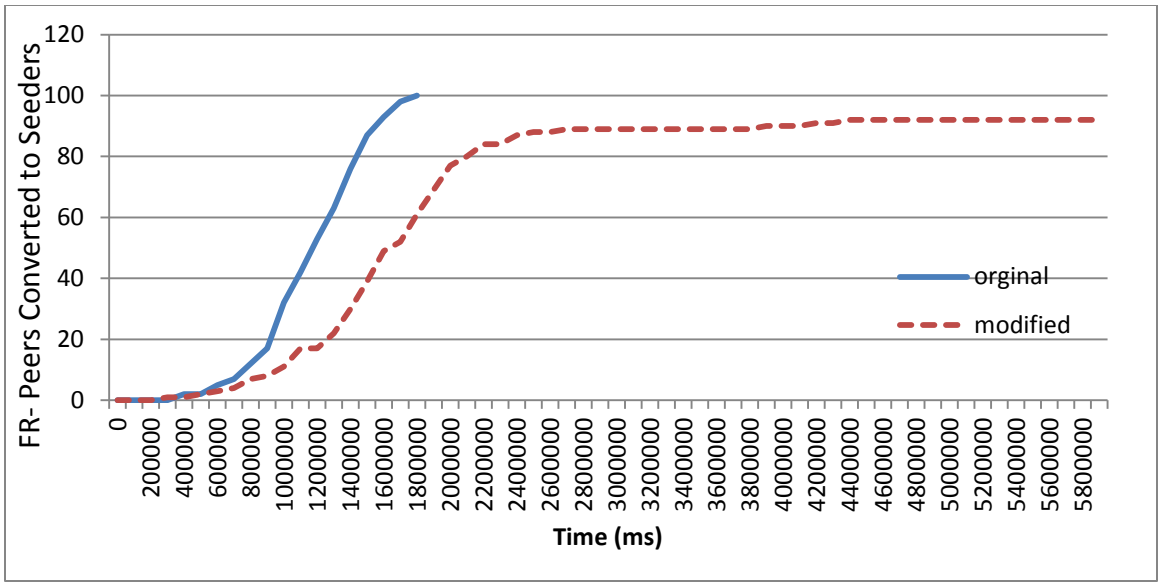


**Figure 49: Transformation of free riders into seeders (1000 peer, 10% free riders, 5% initial seeders)**

| # of Peers | 1000 | % of Free riders | 10 | % of initial seeders | 5 |
|---|---|---|---|---|---|

The same experimental setup was run for an environment with 40% free riders as well. The results show a huge improvement gained by applying the modified choking algorithm. The free riders were delayed sharply and the improvement gained can be quantified by the amount of gap between the blue and the doted red line which represents the transformation rate before and after improving the choking algorithm. Figure 50 illustrates the results.

The same experimental setup was also run for an environment with 70% free riders and the results also show a sharp reduction of the number of free riders who became seeders with time. Although free riders are the majority in this case, the algorithm was smart enough to detect their behavior and slow down their download performance. Figure 51 illustrates the results.

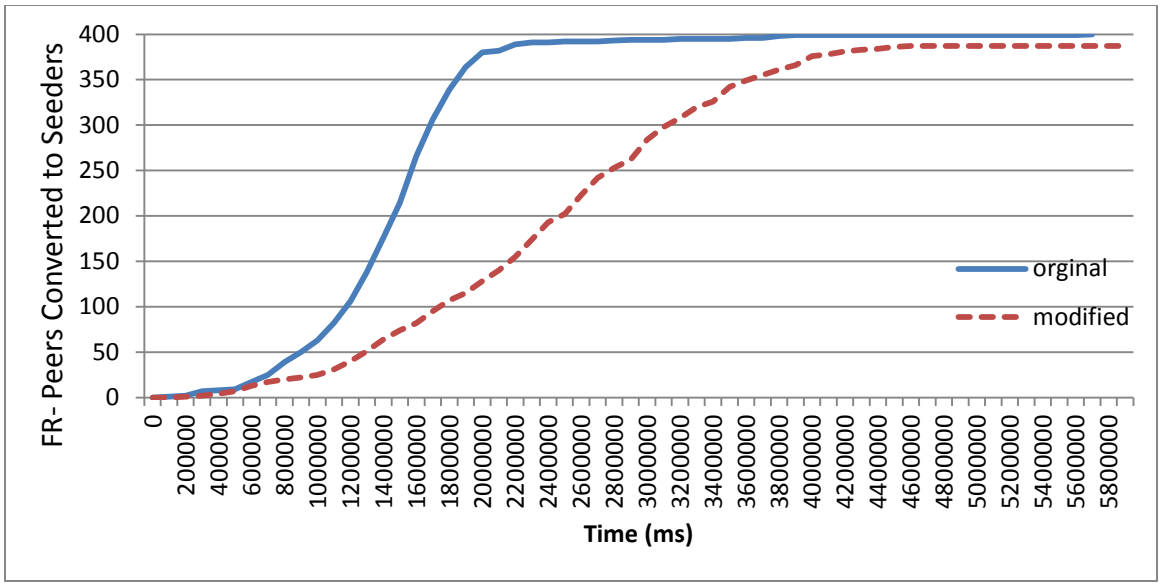


**Figure 50: Transformation of free riders into Seeders (1000 peer, 40% free riders, 5% initial seeders)**

| # of Peers | 1000 | % of Free riders | 40 | % of initial seeders | 5 |
|---|---|---|---|---|---|

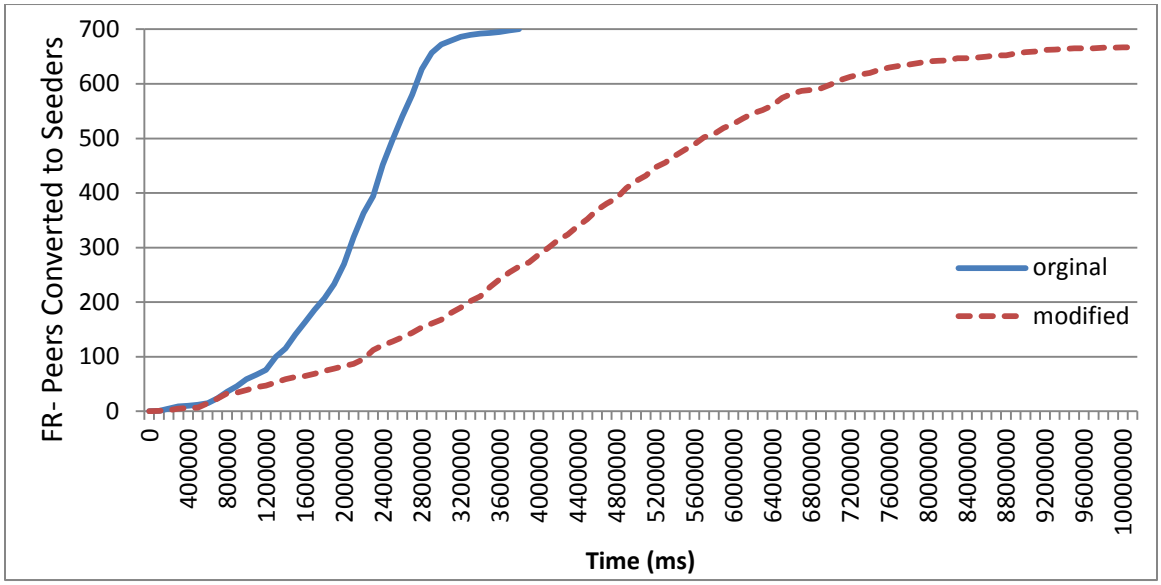


**Figure 51: Transformation of free riders into seeders (1000 peer, 70% free riders, 5% initial seeders)**

| # of Peers | 1000 | % of Free riders | 70 | % of initial seeders | 5 |
|---|---|---|---|---|---|

In addition to the conversion rate from seeder to leechers, the average download completion time for each free rider peer is computed. Figure 52 and 53 shows the average download completion time for free riders an environment with a total of 100 peers, for 8% and 4% initial seeders, for 10%, 40% and 70% free riders.

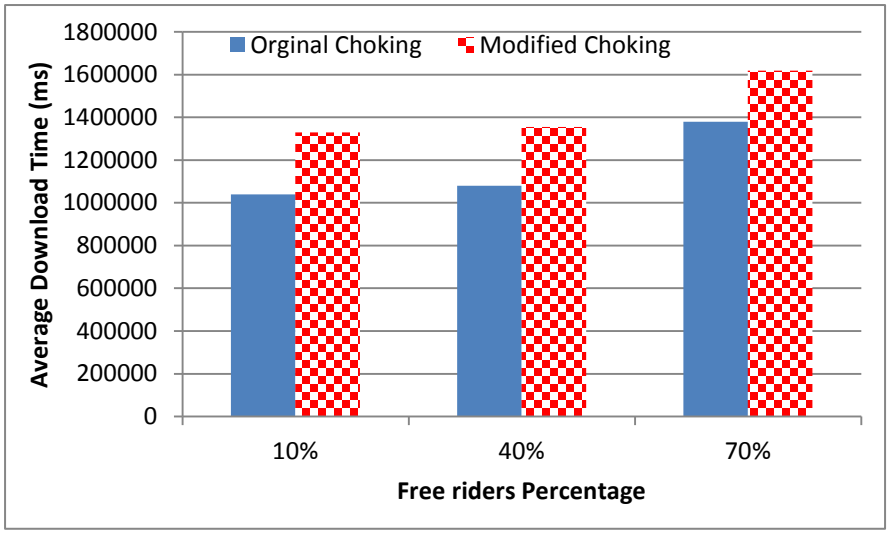


**Figure 52: Average free riders download completion time (8% initial seeders)**

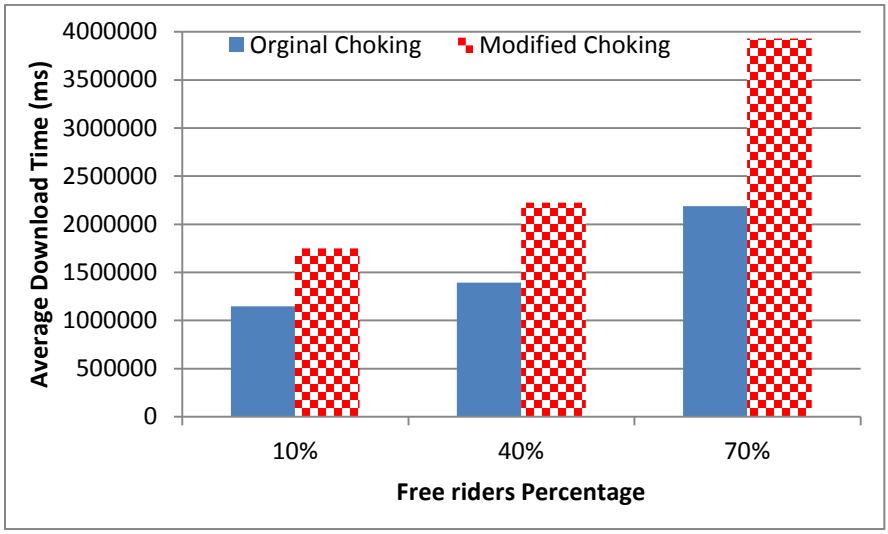


**Figure 53: Average free riders download completion time (4% initial seeders)**

Figure 52 and 53 shows that the modified choking algorithm is more effective when applied to an environment with lower initial seeders. This can be concluded from figure 53. The results also show that as the number of free riders increases, the average download completion time increases too. Comparing figure 52 and 53 also shows that as the number of free riders increases in the environment, the average download time for them increases too. This is due to having lower resources in the environment.

# CHAPTER 6

## Conclusion and Future Work

BitTorrent has been an effective and efficient tool for file sharing. Unfortunately, it is proven that free riding is achievable in BitTorrent. The original choking algorithm of BitTorrent that is utilizing tit-for-tat approach and the randomness in optimistic unchoking are not efficient enough to prevent free riders from taking advantage of the system.

Many studies have shown different achievable attacks to target BitTorrent. An example is BitThief, which can increase the number of connections by sending multiple TRACKER messages to the tracker and get the advantage of connecting to more seeders in the environment. Another example is sending garbage pieces, this is when the free rider sends garbage pieces to trick other who is providing to the system and thus be unchoked normally. Other studies also discussed downloading from seeders only as they do not have any mean of evaluating the contribution of any other peer and this is achievable through sending multiple TRACKER messages to the tracker and contacting seeders only. Another example of client implementation that attacks BitTorrent is BitTyrant that uploads with a minimum speed that allow it to get unchoked. Other attacks also include Sybil attack which is based on accessing the BitTorrent environment with different identities and pieces under reporting strategy attack that tricks the piece-selection strategy which makes the free rider client less interesting in the environment.

Free riding in BitTorrent was mitigated up to a certain extent and many of the discussed attacks were resolved. For example, the hash coding of the pieces and blocks will resolve the problem of garbage uploading. Also, blocking the IP address of the peer who opens multiple connections was resolved in the Azureus BitTorrent implementation. There were also many studies that redesigned the choking algorithm whither in the optimistic unchoking or the regular choking algorithm that occurs every 10 seconds to eliminate free riding.

We have come up with a new way of free riding which is achievable by not sending a HAVE message when a peer complete the download of a specific piece, also free riders will not announce their true file status when they send the BITFIELD message. We proved that free riding is achievable, free riders were normally able to download a piece and they were able to compete with non-free riders without being detected by the choking algorithm.

The choking algorithm, especially when run by a seeder is not effective as the seeder has no mechanism to detect the free riding behavior as the seeder is not asking for any piece. Leechers will be able to reduce the effect of free riders in the 10 seconds choking algorithm as they will sort by the contribution of their neighbors and pick the top contributors to be unchoked. The optimistic unchoking algorithm is not effective in detecting free riders for both seeders and leechers.

We have come up with an approach that improves the choking algorithm in the leechers and the seeders who used to be leechers , we call them seeders with experience. This experience is gained by interacting with others and asking for pieces before becoming

seeders. Our approach depends on chance giving as each leecher and seeder who used to be a leecher will give X number of chances of free pieces upload. After X pieces are uploaded to a given peer, it will not be unchoked again unless it starts providing to the environment.

Our approach was tested with multiple experimental setups and demonstrated that free riders will be punished and their file download performance will be delayed compared with non-free riders and sometimes free riders will never get the complete file and that depends on their swarm.

We believe that our approach does not introduce any negative impact to the BitTorrent Environment. Our approach is also resistance to free riding attacks that are considered in the original BitTorrent protocol. For example, uploading garbage will not work as there are ways to immediately validate received blocks, and punish their sender by blocking its IP address. Also peers cannot form a group and attack our algorithm as the peer selection is random. In addition, there is a chance for dishonest peers to cheat the system, as our approach does not depend on recommendation or voting techniques.

Our approach also maintains the original objectives of BitTorrent choking algorithm. Leechers will still sort their neighbors by their contribution to them. Seeders will still sort their neighbors by how much they have previously received from them and this will maintain the objective of producing more seeders in the environment as soon as possible. The optimistic unchoking is still based on randomness which will give a chance to new comers to be unchoked as long as they contribute to the environment in the near future and this is controlled by the X chances giving technique.

The future work to improve our algorithm is to find an approach for original seeders to mitigate free riders and this is achievable through different mechanisms such as recommendations from other leechers in the original seeder swarm or hiding the identity of original seeders and allow them to ask piece that they do not need as a way to test their neighbors. We can also implement other attacks discussed in the literature and prove that our algorithm is still resilience to them. In addition, all the experiments run are with a static environment consisting with a fixed number of peers that they do not leave the system. As a future work, we will test our approach with a dynamic environment where peers can leave the environment and new peers can join at different stages of the simulation.

The selection of the X number of blocks need also to be tested more, and also it could be selected through another algorithm based on the bandwidth of each peer and the performance of its neighbors. A low X number can be used with fast peers while a higher one can be used with slow peers. A low X number will lead to more punishment to free riders while it could also negatively affect non-free riders who are slow or did not have a chance to upload any block at an early time. A large X number will allow more free riders but will give more chance for slow non-free riders to be unchoked.

# Bibliography


[1] Atlidakis V. (2014). EnhancedBit: Unleashing the potential of the unchoking policy in the BitTorrent protocol. *Journal of Parallel and Distributed Computing*, pp.1959–1970.

[2] Bharambe A., Herley C., Padmanabhan V. (2006). *Analyzing and Improving a BitTorrent Networks Performance Mechanisms* . Barcelona: 25th IEEE International Conference on Computer Communications.

[3] Biazzini M., Serrano-Alvarado P., Carvajal-Gomez R. (2013). Towards improving user satisfaction in decentralized P2P networks. *2013 9th International Conference. Collaborative Computing: Networking, Applications and Worksharing (Collaboratecom)* (pp. 315 - 324). Austin, TX: IEEE.

[4] *BitTorrent Official website*. (n.d.). Retrieved from http://www.bittorrent.com/company/about

[5] Cohen B. (2003). Incentives Build Robustness in BitTorrent. *ACM* (1-59593-026-4/05/0008). Philadelphia,: bitconjurer.org.

[6] Frioli M. (2008). *A BitTorrent module for Peersim.* PeerSim-http://peersim.sourceforge.net.

[7] Ge T., Manoharan S. (2010). Mitigating Free riding on BitTorrent Networks. *2010 Fifth International Conference on Digital Telecommunications (ICDT)* (pp. 52 - 56). Athens: IEEE.

[8] Izhak-Ratzin R., Park H., van der Schaar M. (2011). Reinforcement learning in BitTorrent systems. *INFOCOM, 2011 Proceedings IEEE* (pp. 406 - 410). IEEE.

[9] Karakaya M., Korpeoglu I., Ulusoy O. (2009). Free Riding in Peer-to-Peer Networks. *Internet Computing, IEEE*, pp. 92 - 98.

[10]Levin D, LaCurts K., Spring N., Bhattacharjee B. (2008). BitTorrent is an Auction:Analyzing and Improving BitTorrent's Incentives. *SIGCOMM 2008 conference on Data communication* (pp. 243-254). New York: ACM.

[11] Li M., Yu, J., Wu J. (2008). Free riding on BitTorrent-Like Peer-to-Peer File Sharing Systems: Modeling Analysis and Improvement. *IEEE-Parallel and Distributed Systems*, pp. 954-966.

[12] Liogkas N, Nelson R., Kohler E., Zhang, L. (2006). *Exploiting BitTorrent For Fun (But Not Profit).* Los Angeles: The 5th International Workshop on Peer-to-Peer Systems (IPTPS).

[13] Moor P. (2006). Free Riding in BitTorrent and Countermeasures. *Master Thesis*. Swiss Federal Institute of Technology Zurich.

[14] Paolo D, Cucchiella S. (2009). Selfish strategies affecting the BitTorrent protocol. Vasteras: Malardalen Univeristy.

[15] Rafat I, Park H., van der Schaar, M. (2012). Online Learning in BitTorrent Systems. *IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS*, pp. 2280-2288.

[16] Raymond L, Muppala J. (2010). A Survey of BitTorrent Performance. *IEEE COMMUNICATIONS SURVEYS & TUTORIALS*, pp. 140 - 158.

[17] Sharma A. , Bhakuni A., Kaushal R. (2013). Performance analysis of BitTorrent protocol. *National Conference on Communications (NCC)* (pp. 1 - 5). IEEE.

[18] Sherman A., Stavrou A., Nieh, J., Stein C. (2008). *Mitigating the Effect of Free riders in BitTorrent using Trusted Agents.* New Yourk: Department of Computer Science, Columbia University.

[19] Sirivianos M., Han J., Rex P., Yang C. (2007). Free riding in BitTorrent Networks with the Large View Exploit. *IPTPS.* Bellevaur: Microsof Research.

[20] Stevens L., Legout A., Dabbous W. (2010). Pushing BitTorrent locality to the limit. *Computer Networks*, pp. 541–557.

[21] Wan J., Shen R., Ullrich C., Luo H., Niu C. (2010). Resisting free riding behavior in BitTorrent. *Future Generation Computer Systems 26*, pp. 1285-1299.

[22]Naldi M., D'Acquisto G. (2013). When free riding is the best choice: The case of network charges for content providers. International Conference on Network and Service Management (CNSM) (pp. 304 - 309). Zurich : IEEE

# Vitae

| | |
|---|---|
| Name | Murtada Ibrahim Al-Habib |
| Nationality | Saudi |
| Date of Birth | 3/13/1984 |
| Email | habemi0a@aramco.com |
| Address | Engineering Office Building - Saudi Aramco- Dhahran |
| Academic Background | • BS in Information and Computer Science, KFUPM-Dhahran, Saudi Arabia<br>• MS in Subsurface Geoscience, Rice University-Texas, U.S.A |
| Career | • Exploration Systems Analyst, Saudi Aramco (2007 – present)<br>• Internship at LandMark- Halliburton (Jun 2011, January, 2012 ) |
| Certificates | • Sun Certified Java Developer<br>• Sun Certified Web Components Developer<br>• Best COOP student, 2006, Saudi Aramco |