2010

No space left behind

Joseph Imbasciano

ROCHESTER INSTITUTE OF TECHNOLOGY

No Space Left Behind

Joseph Imbasciano
September 2010

Thesis submitted in partial fulfillment of the requirements for the
degree of Master of Science in
Networking and Systems Administration

B. Thomas Golisano College of
Computing and Information Sciences
School of Informatics
Department of Networking, Security, and Systems Administration

Rochester Institute of Technology

B. Thomas Golisano College
of
Computing and Information Sciences

Master of Science in
Networking and System Administration

Thesis Approval Form

Student Name: Joseph Imbasciano

Thesis Title: No Space Left Behind

Thesis Committee

Name	Signature	Date

Professor George Barido
Chair

Dr. Yin Pan
Committee Member

Jason Koppe
Committee Member

Thesis Reproduction Permission Form

Rochester Institute of Technology

B. Thomas Golisano College
of
Computing and Information Sciences

Master of Science in
Networking and System Administration

No Space Left Behind

Date: 9/15/2010 Signature of Author: _____

Abstract

A unique storage problem occurred in the College of Science at RIT. A need for additional centralized storage was needed, but there were no additional resources allocated for centralized infrastructure. However, there was an abundance of free disk space on the lab computers that existed within the college. After calculating the total amount of hard disk space for each lab, it was found that there was approximately 36 terabytes of unused disk space per lab. The challenge was how does one utilize this space, yet still maintain reliability and availability of files?

This thesis describes the development of a proof of concept system which utilizes free storage space on lab computers within the college. The system automatically distributes a file to enough computers so that it may be retrieved even in the case of computer failures. It also is able to periodically determine whether or not the file is needed on additional computers within the system in order to maintain availability of the file.

The proof of concept system was implemented and tested to demonstrate the effect on individual clients within the system. Results have shown that while it has little effect on CPU and RAM utilization, it does have the ability to utilize all available network bandwidth.

Acknowledgements

I would like to thank the following people who helped make this thesis possible:

Professor George Barido, for his support and guidance throughout the entire development of this thesis and for always challenging me throughout my career at RIT.

Dr. Yin Pan, for her help and feedback during the writing of this thesis and for always providing fun and interesting classes.

Jason Koppe, for always offering his help and ideas, both on this thesis as well as in the lab.

Adrian Gibrat, for publishing the Torrent.php class used within this thesis.

My friends, for keeping life interesting and teaching me things I never knew I wanted to learn about.

My family, for their love, support, and enthusiasm throughout my entire RIT career.

Table of Contents

Appendix

List of Tables

List of Figures

1

Introduction

1.1 Background

A unique storage problem occurred in the College of Science at RIT. The problem was that the department was running out of centralized storage space for servers, backups, and software. Due to budget constraints the college was unable to purchase additional centralized storage, but it was allotted money for new lab computers. Each of the new computers came equipped with a one terabyte hard drive. After calculating the used space on each drive it was found that there was approximately 900 gigabytes of free space per computer. Each lab consisted of 40 computers, meaning that one lab housed around 36 terabytes of free disk space. The computers were almost always available for use, except when software updates and new images were pushed out.

The question became how does one use the free disk space but account for hardware failure, outages, and occasional imaging? This thesis aims to solve the problem that was presented in the College of Science by using existing technologies in new ways.

1.2 Executive Overview of Solution

In order to provide for high availability of files one must replicate files to different computers. In this situation not only must these files be replicated among computers, but also dispersed among different computing labs. Replicating to multiple computers will account for occasional hardware failures, and imaging a single machine. Replication to multiple computing labs will account for entire lab outages in the case of network failure, power outages, or entire lab upgrades.

The BitTorrent protocol was chosen as a foundation for the proof of concept solution. By utilizing BitTorrent one is able to distribute files to multiple hosts. Even if one of these hosts were to become unavailable the file would still be able to be retrieved as other hosts on the network would still carry a full copy of the file.

A system was built upon the BitTorrent protocol which allows a user to choose a file to distribute into the environment. It allows one to choose the number of computers to distribute the file to and the number of computing labs that the file should exist in. From this, the system automatically distributes the file to the appropriate number of computers and computing labs. Periodically the system checks to ensure there are enough computers with the file, and also if the file is distributed to enough computing labs. If it detects that there is a shortage it will appropriately react and distribute the file to additional hosts as necessary.

2

Related Work

Researchers Butt et al. have previously looked into enhancing the Network File System (NFS) with a peer-to-peer backend [2]. They were able to build a system termed Kosha which utilized cluster nodes in order to provide additional redundancy, storage, and load balancing that NFS could not do alone. The file system used a series of virtual directories which are seen as regular NFS mount points to the client. From this a centralized server managed the replication of the files across the peer-to-peer network. The end result was a distributed NFS file system which spanned across multiple nodes using a peer-to-peer backend. Additionally this file system was still accessible through NFS protocols and had the ability to use NFS permissions. Stein et al. [5] took a similar approach by utilizing a peer-to-peer backbone of resources, and then built upon the NFS file system in order to use features from both technologies. The peer-to-peer technology was robust and provided a higher level of reliability, while the NFS file system provided permissions and access controls.

The fundamental difference between these researchers' approach and the proof of concept model built for this thesis was that there was an underlying dependence upon the network file system. The goal of their projects was to build peer-to-peer functionality into the network file system. The goal of this thesis was to build a platform that allowed for files to be distributed across multiple hosts. Future development of the project could allow for more compatibility across different host operating systems, allowing for a larger pool of potential storage.

Other peer-to-peer backup systems have also been looked into such as the system PeerStore [4]. Researchers Landers et al. have demonstrated a system which is used primarily for peer-to-peer backup purposes. The basic concept was that PeerStore allowed a group of users to store backups on others' computers. Through this system a user could backup his or her files to the group and recover them as needed. A metric for fairness and ratios was implemented to reduce the chance of abuse. The system aimed to be decentralized so that clients negotiate among each other rather than through a centralized system.

A similar service is the peer-to-peer storage system developed by Boian and Boian which aimed to provide additional storage through a web service and peer-to-peer storage [1]. The system introduced a custom kernel module FUSE which is used in conjunction with a web service to enable peer-to-peer communication. The goal of this project was to present a user with a means of storage through peer-to-peer communities that they join. This project also attempted to decentralize the means of storage decreasing the chance of failure and increasing availability. In addition it aimed to build a file system that would work with the host itself through the FUSE kernel module.

3

Methodology

3.1 Concept

3.1.1 Key Technologies

BitTorrent

BitTorrent is a peer-to-peer file sharing protocol which is capable of distributing large amounts of data between clients from multiple sources simultaneously. Clients split the file into many smaller parts and then transfer each of these smaller parts. This enables clients to download and share parts of the file even if they have not yet completed the transfer. Clients that have 100% of the file are referred to as seeds.

BitTorrent Tracker

A centralized tracker may or may not be used depending on the implementation of the torrent. The tracker is used in order to tell clients about other clients who wish to download a file. It is capable of tracking the health of the file as well, calculating the ratio of seeders to the number of clients who wish to download (leeches).

Framework

A framework is something used in software development which allows you to extend on the basic functionality that is provided by it. Currently many different frameworks exist to develop customized BitTorrent trackers. An existing BitTorrent tracker framework is used in this implementation in order to simplify development.

Authentication Framework

An authentication framework is similar to a framework with the exception that it deals specifically with authentication. The BitTorrent tracker framework used in this thesis includes a simple authentication framework which is used to authenticate a single user to the tracker. This user has access to create new files in the system, as well as retrieve files from the system.

3.1.2 Implementation

A user accesses the client end of the system. He or she has the ability to upload or download files from the interface and is not responsible for the initial configuration of the system.

The first step is to access the add file interface which can be seen in figure 3.1 below. Here the user is able to choose the file they wish to upload, as well as configure the number of zones and seeds/zone they wish the file to reside in. After submitted, the file will be uploaded to the tracker and seeded out to the appropriate number of computers. Periodically the system will query the tracker in order to determine whether or not the file is present on enough seeds and in enough zones. If it is not, the tracker will appropriately instruct additional seeds to download the file.

A logical topology of the entire system can be seen in figure 3.2. This demonstrates how computers are split into logical zones for which the tracker can choose to seed to. It can be seen that there are four logical zones with four seeds per zone active.

Figure 3.3 shows a logical representation of the process that occurs. For example: The user configures the file to be available on two seeds per zone and to exist in two zones. The tracker in this instance tells computers A and B in zones 1 and 2 to download the file (Figure 3.3-1). If computer A in zone 1 were to become unavailable, the tracker would instruct computer C in zone 1 to download the file from the computers marked as seeds (Figure 3.3-2). If zone 1 were to become entirely unavailable, computers within zone 3 would then be instructed to download and seed the file and so on (Figure 3.3-3).

Through this process files are able to be distributed and replicated in order to maintain availability. When a user wishes to access a file stored within the system they authenticate to the tracker interface, and download the torrent file via a BitTorrent client. The BitTorrent client will allow the user to access the file from all of the systems simultaneously, increasing download speed and decreasing the chance of failure.

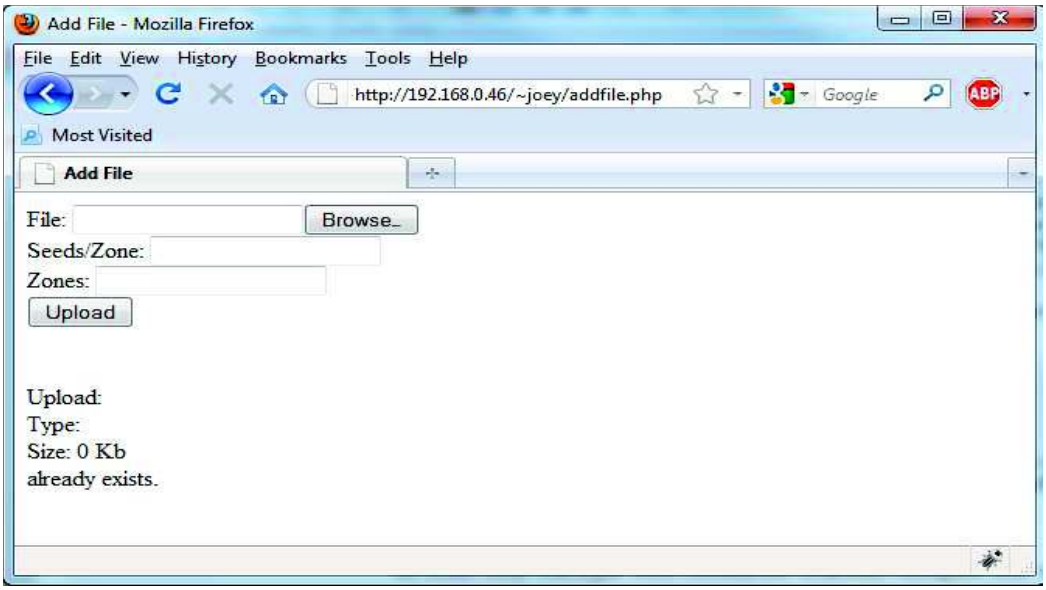


Figure 3.1: Add File Interface

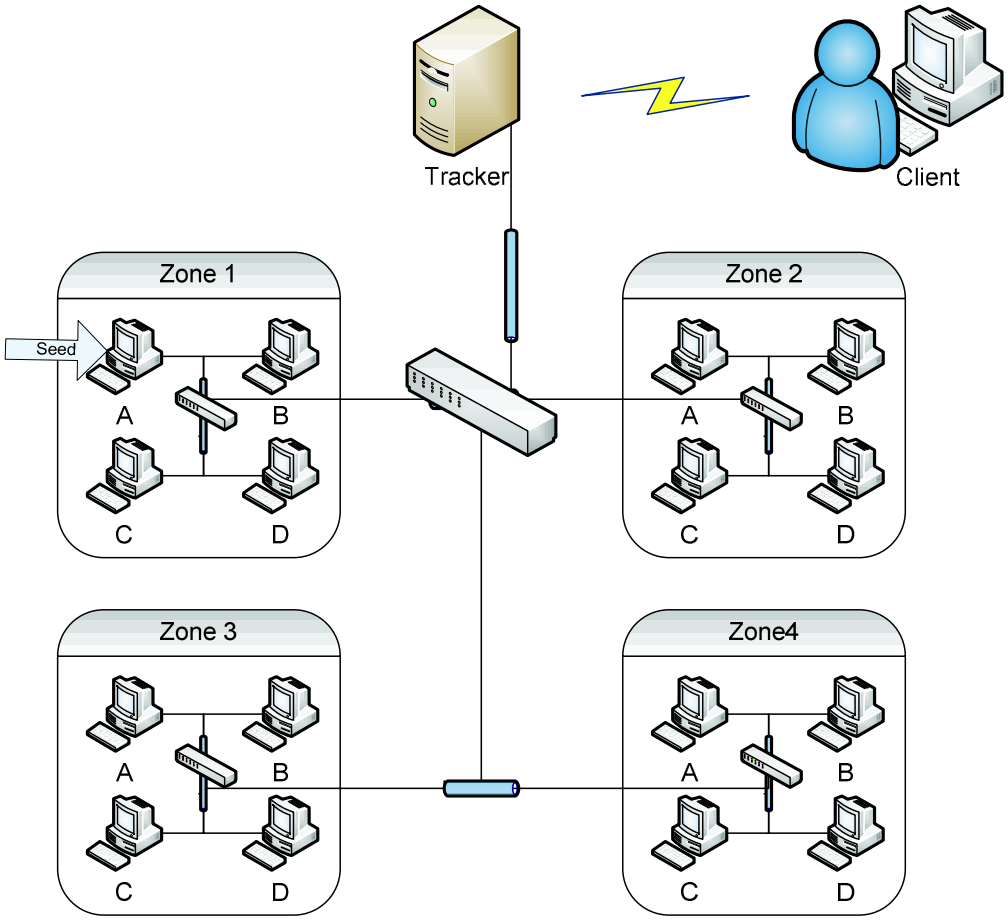


Figure 3.2: Logical Topology of System

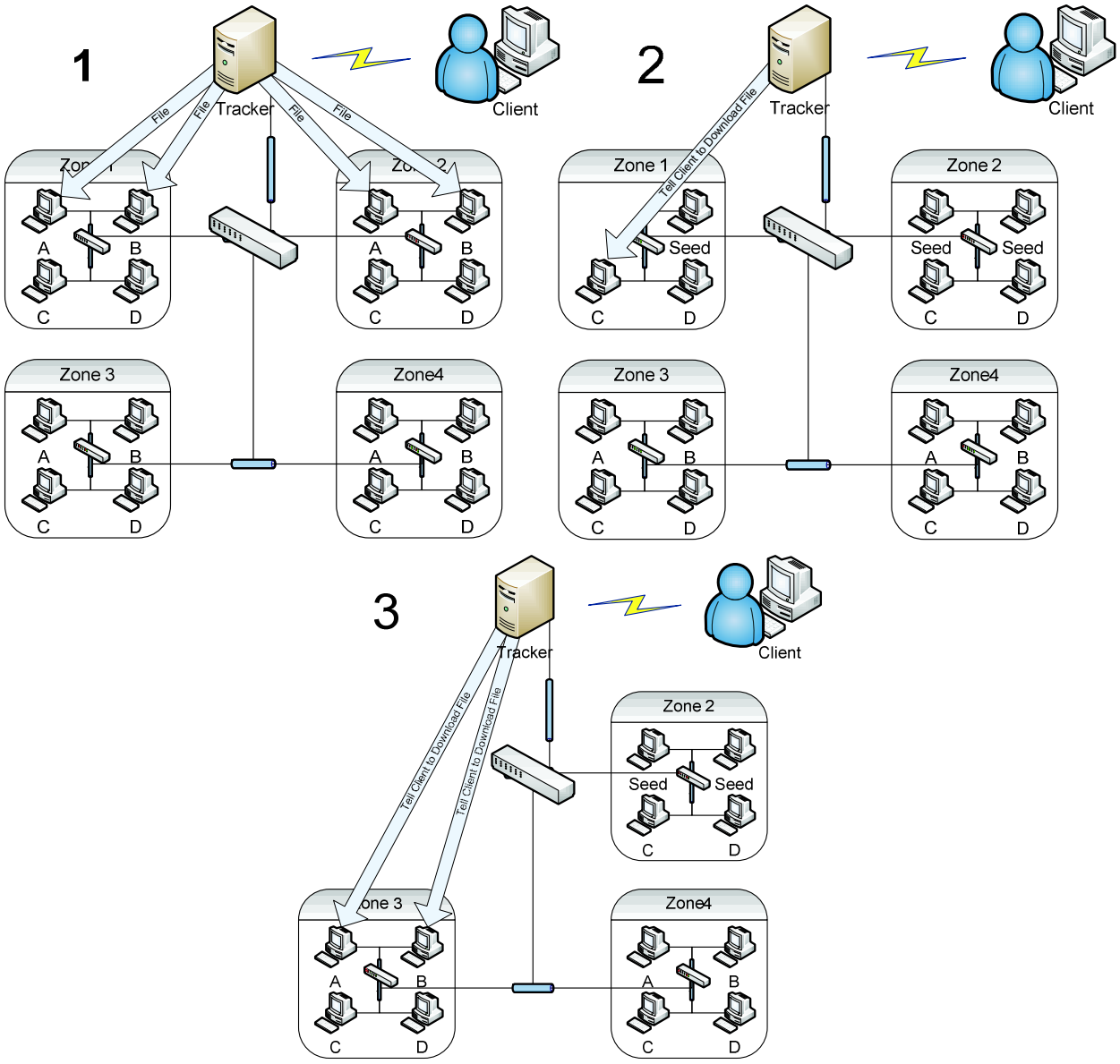


Figure 3.3: Replication Process

3.1.3 Recommended Values

The number of seeds/zone and number of zones are configurable options by the user. The minimum recommended number of seeds/zone is 2. By using at least 2 seeds/zone this ensures that if there is a hardware failure of a computer in the zone, another computer will still carry a full copy of the file. Similarly, 2 is the recommended minimum value for the number of zones. This is to ensure that if one zone were to become entirely unavailable due to network or power outage, another zone would still carry a full copy of the file.

3.1.4 Use Only in a Secured Environment

The proof of concept system described and documented in this thesis is intended for use only in a secured environment. Security is outside of the scope of this thesis; therefore one should take caution before implementing the proof of concept in his or her own environment.

3.2 Conventions Used

1. <Variable> - Words contained within <> represent a variable value.
    a. Example: http://192.168.0.46/~joey/rss/<IP>/rss.xml
        i. Where <IP> represents an IP address.

2. *Table name* - *Italicized* words represent MySQL table names.
    a. Example: *peers table*

3.3 Environment Design

3.3.1 Server

The server was built within VMware Workstation 6.5.1 build-126130. The reason for building the server in a virtual environment is that it is much more portable. One is able to build the server at home, and then transport it to the testing environment easily. Table 3.1 describes the physical machine that the server was run upon. Table 3.2 describes the server itself; both virtual machine specifications, and software versions.

Component	Specification
Operating System	Windows 7 x64 Enterprise (6.1.7600)
VMware Workstation	6.5.1 build-126130
CPU	Core i5-750 @ 2.66GHz
Memory	4GB Dual Channel
Hard Drive	500GB 7200RPM WD
Video	Nvidia GeForce GTS 250

Table 3.1: Physical Machine Specifications

Component	Specification
Debian Linux	5.0.4, Kernel 2.6.26-2-686
Apache	2.2.9 (Debian)
PHP	5.2.6-1+lenny8
MySQL	5.0.51a-24+lenny4
phpMyAdmin	2.11.8.1deb5+lenny4
CPU	1 Processor
RAM	2048 Megabytes
Hard Drive	50 Gigabytes
NIC	Bridged

Table 3.2: Virtual Machine Specifications

All software was installed using Debian's package management system apt-get. The following additional configuration was done:

Debian

- eth0 was configured for a static IP address 192.168.0.46.

Apache

- The module libapache2-mod-php5 was enabled. This allows for PHP scripts to run via the apache server.
- User directories were enabled, allowing for the following URLs to map to the users:
    - http://192.168.0.46/~joey/ **maps to** /home/joey/public_html/
    - http://192.168.0.46/~rivet/ **maps to** /home/rivet/public_html/

PHP

The following changes were made to the default php.ini configuration:

Name	Original Value	Updated Value	Purpose
memory_limit	8M	1024M	Maximum amount of memory a script may consume.
max_execution_time	30	600	Maximum execution time of each script (seconds).
max_input_time	60	600	Maximum time script may parse a request (seconds).
post_max_size	8M	1000M	Maximum size of POST data PHP will accept.
file_uploads	Off	On	Allow for file uploads.
upload_max_filesize	2M	1000M	Maximum allowed size for file uploads.

Table 3.3: PHP Configuration Changes

These changes were made in order to allow a file up to 1000 megabytes to be uploaded to the server via a POST[1] request.

- memory_limit – Increased to 1024MB to allow for enough memory to store a 1000MB file being uploaded. May be able to be decreased depending on environment.
- max_execution_time, max_input_time – These values increased in order to prevent script timeouts from large file uploads. These times may be able to be decreased depending on environment.
- post_max_size, upload_max_filesize – Increased in order to allow for 1000MB files to be uploaded.
- file_uploads – Changed to on to allow for file uploads.

[1] A POST request is an HTTP request which submits form data in the body of an HTTP request.

3.3.2 Clients

Sixteen physical clients were used in this experiment consisting of the following configuration:

Component	Specification
Operating System	Windows 7 x64 Enterprise (6.1.7600)
BitTorrent Client	uTorrent 2.0.2 (build 19648)
CPU	Core i5-750 @ 2.66GHz
Memory	4GB Dual Channel
Hard Drive	500GB 7200RPM WD
Video	Nvidia GeForce GTS 250

Table 3.4: Client Configuration (Physical Machines)

All software was installed with the default configuration. The following modifications were made to each client:

Windows 7

The following services were disabled:

- Windows Firewall – Disabled to allow all connections in and out from all clients within system.
- Windows Updates – Prevent windows update service from consuming network bandwidth.
- Windows Defender – Disabled in order to prevent resources from being consumed by scans of files.

uTorrent

Each uTorrent client was manually configured to point to the RSS feed on the server that controlled the respective client. The RSS URL was configured as follows:

- http://192.168.0.46/~joey/rss/<IP>/rss.xml
    - Where <IP> is the IP address of the client.

3.3.3 Network

A Cisco 2950 24 port switch, running IOS 12.1 (22), was utilized in the testing of the proof of concept model. The switch was configured so that all ports were on the same VLAN.

3.4 Application Design

3.4.1 Overview

Figure 3.4 describes an overview of the system from a general view. There are two major components to the system which interact with each other. The first component is the Rivet Tracker which is in charge of managing the BitTorrent tracker. This system is comprised of a BitTorrent tracker written in PHP which stores information in a MySQL database referred to as the Rivet database. There are two interfaces to this system, the first is through the web interface, and the second is directly through the MySQL database.

The second component is the Master which is in charge of the entire system as a whole. This component is the only one that the user interacts with. The purpose of this component is to run the PHP scripts which create and maintain the replication process. The clients are controlled from the Master via RSS feeds. The Master stores information in a separate database referred to as the Master database.

The following process takes place when a file is added:

1. The user adds a file to the system via the Master add file script.
2. The Master updates the Master database with the file information.
3. The Master interacts with Rivet Tracker and adds the file to the tracker via web interface.
4. The Master instructs the appropriate clients to download the file via RSS feeds.
5. Periodically the Master queries the rivet database in order to ensure each file is present in enough seeds and zones.

For the proof of concept built for this thesis, each of the separate components resides on a single server.

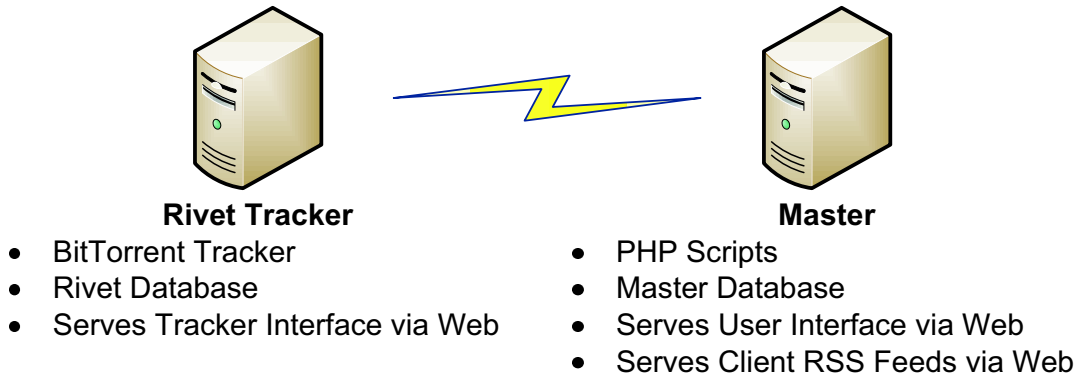


Figure 3.4: General Logical View of System

3.4.2 File Structure

In order to gain a better understanding of the system implemented in this thesis an overview of the file structure is needed. Table 3.5 below describes the layout of the file structure used.

Path	Purpose	Web URL
/home/rivet/public_html/	Contains installation of Rivet Tracker.	http://192.168.0.46/~rivet/
/home/joey/public_html/	Contains scripts and directories used for replication system.	http://192.168.0.46/~joey/
/home/joey/public_html/rss/	Contains directories of client IPs connected to the system.	http://192.168.0.46/~joey/rss/
/home/joey/public_html/rss/<IP>/	Each client used within the replication system has a unique IP. For each of these IPs a directory is used to store the rss.xml file for the client.	http://192.168.0.46/~joey/rss/<IP>/
/home/joey/public_html/cookies/	Cookies used by the system are stored within this directory.	http://192.168.0.46/~joey/cookies/
/home/joey/public_html/torrents/	Torrent files created for each uploaded file are stored here.	http://192.168.0.46/~joey/torrents/
/home/joey/public_html/uploads/	Location where uploaded files are stored for initial seeding.	http://192.168.0.46/~joey/uploads/

Table 3.5: File Structure of System

3.4.3 Database Structure

Two separate databases were used in the creation of the proof of concept for this thesis. The first is the Rivet database, the second is the Master database.

Rivet Database

While there are many tables that the BitTorrent tracker uses, only the ones relevant to the function of the Master component will be described below.

namemap table

This table is used by the Master in order to determine the proper filename and size of a file when it is queried from the system. The info_hash field is used to match file references between all of the databases used. Table 3.6 describes the fields, types and purpose used in the namemap table.

Field	Type	Purpose
info_hash	char(40)	Hash of file. Used to uniquely identify a file.
Filename	varchar(250)	Name of file.
url	varchar(250)	Unused.
Size	bigint(20)	Size of file in bytes.
pubDate	varchar(25)	Date file added to tracker.

Table 3.6: Overview of namemap table used in Rivet database.

x<info_hash> table

For each file that is added to the tracker, it creates a new table which is named based on the info_hash for the file. The info_hash is a 20 byte sha1 hash of the torrent metainfo which is contained in the .torrent file created for each file. Each of these tables starts with the letter x and is then followed by the info_hash for the file, an example being: x4f7117e3100ff645d940f0d5ed76164d4d177878. Therefore the tracker creates a tabled named x<info_hash> for each file present. Each of these tables contains the same format which is shown in table 3.7.

These tables are used to determine which peers currently are seeding the file. If a peer is seeding the file this means that they have 100% of the file and it can be assumed that the file has successfully replicated to this peer. Only two fields are relevant to the Master system, the IP and the status fields.

Field	Type	Purpose
peer_id	char(40)	Unique peer hash.
bytes	bigint(20)	Bytes transferred to peer.
ip	char(50)	IP address of peer.
port	smallint(5)	Port used by peer for transfer.
status	enum('leecher', 'seeder')	Tells whether the peer is a leecher or seeder.
lastupdate	int(10)	When the information was last updated.
sequence	int(10)	Used internally by tracker.
natuser	enum('N', 'Y')	Whether or not the peer is behind a NAT.

Table 3.7: Overview of x<info_hash> table used in Rivet database.

Master Database

The Master database is used by the PHP scripts written for the system in order to store and query data as needed. There are two main tables named *files* and *peers*. For every peer that is added to the system a table is added which is named after the IP of the peer.

files table

This table stores a list of files which have been added to the system. If a file is present in this list it will be replicated to the appropriate number of peers based on what was configured. It is important to note that the file_id field matches the info_hash field described in the Rivet database. Table 3.8 describes the table layout.

Field	Type	Purpose
filename	varchar(255)	Filename of file.
file_id	varchar(255)	Hash of file. (Matches info_hash in Rivet Database)
seeds	int(11)	Number of seeds per zone that should be present.
zones	int(11)	Number of zones file should be present in.

Table 3.8: Overview of *files table* used in Master database.

peers table

This table is responsible for storing the list of peers which are present in the system. These peers must have a BitTorrent client installed and configured to point to their respective RSS feed on the Master. This database is manually populated by an add peer script. Table 3.9 describes the table layout.

Field	Type	Purpose
ip	varchar(15)	IP address of peer.
zone	int(11)	Zone which peer is located in.
space	int(11)	Space in megabytes available for files on peer.

Table 3.9: Overview of *peers table* used in Master database.

<IP> table

For each IP that is added to the peers table a corresponding table is added for the IP. For example if the IP 192.168.0.1 was added to the peers table the table 192-168-0-1 is created. These tables are used in order to store which files should be present on each of the peers. The RSS feeds which are used to control the peers read these tables and generate the RSS feed based on the files present in the table. Table 3.10 describes the layout of these tables.

Field	Type	Purpose
filename	varchar(255)	Filename of the file.
file_id	varchar(255)	Hash of file. (Matches info_hash in Rivet Database)

Table 3.10: Overview of *<IP> table* used in Master database.

3.4.4 Initial Configuration of System

Since the thesis was a proof of concept model only, much manual configuration of the system was done in order to get things working properly. This section describes the steps taken on the server, clients, and within the system itself which must me manually done.

Server

The directory structure described in 3.3.2 must be manually created. Additionally the following permissions must be set on directories as described in table 3.11:

Path	Permissions
/home/joey/public_html/rss/	**777**
/home/joey/public_html/cookies/	**777**
/home/joey/public_html/torrents/	**777**
/home/joey/public_html/uploads/	**777**

Table 3.11: Permissions manually set for directories.

These permissions are necessary so that the PHP scripts and Apache server may write and execute from these directories.

BitTorrent System

- The Master database *files table* and *peers table* described in 3.3.3 must be manually created.
- The *peers table* must be manually populated through the addpeer.php script.

Clients

Each of the clients that one wishes to add to the system must be manually configured. The BitTorrent client must be manually installed and configured to automatically download files from the RSS feed which relates to the client. An example of this can be seen in figure 3.5 in which the client's IP address is 192.168.0.50. The RSS feed that the client must point to is:

- http://192.168.0.46/~joey/rss/192.168.0.50/rss.xml

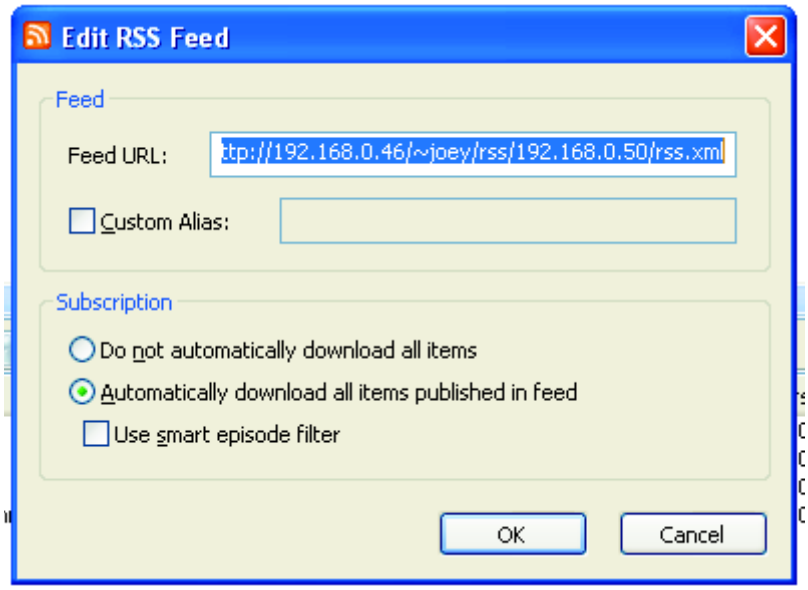


Figure 3.5: Manual configuration of RSS feed on BitTorrent client.

3.4.5 Flow of System

This section describes the flow of the system in detail showing at a detailed level what is happening within the entire system. It is assumed at this point that initial configuration has been done and the system is ready for use.

Adding a File

1. The user visits the URL http://192.168.0.46/~joey/addfile.php (Figure 3.6)
    a. Chooses file to upload.
    b. Sets the number of seeds per zone file should be present in.
    c. Sets the number of zones the file should be present in.
    d. Clicks upload.
2. The file is uploaded to the server using a POST request.
3. File is moved from temporary location to /home/joey/public_html/uploads/
4. A torrent file is created for the file which contains the metadata necessary for adding the file to the BitTorrent tracker.
5. The file is added to the BitTorrent tracker using the torrent file created.
    a. This step executes an external script tracker_interface.php.
        i. Using a cURL[2] query this script authenticates with the tracker via the web interface.
            1. The cookie for this authentication is stored in /home/joey/public_html/cookies/ .
        ii. A second cURL query is made in order to add the torrent to the tracker.
6. A MySQL query is executed updating the *files* table in the Master database.
    a. INSERT into files (filename, file_id, seeds, zones)
7. Script returns success or failure.

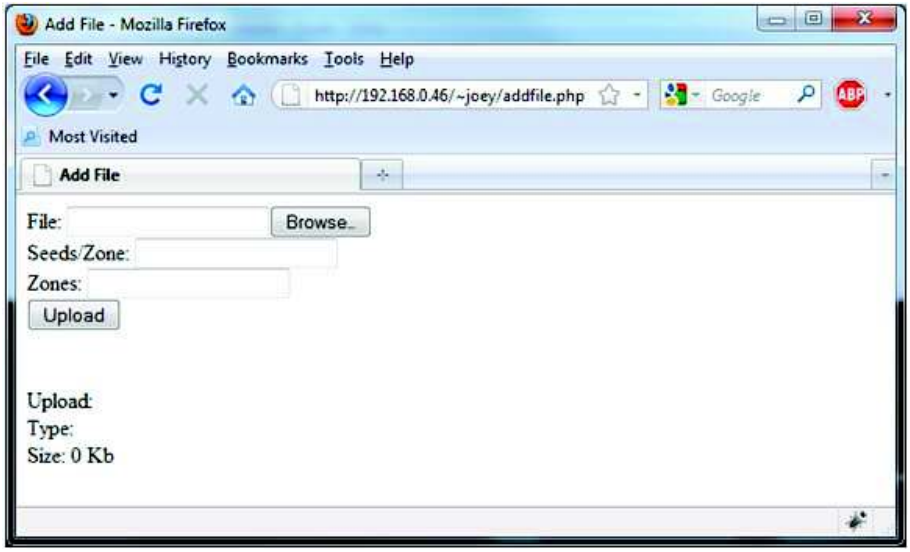


Figure 3.6: Adding a file to the system via addfile.php.

[2] cURL is a library implemented in PHP that allows for HTTP connections and communication.

Updating File Status & Seeding Files to Clients

Once a file is uploaded a script, refresh_all.php, executes once every 15 minutes via a cronjob. This ensures that all files within the *files table* have enough seeds per zone and are present in enough zones. The following takes place when this script is executed:

refresh_all.php

1. MySQL query executed selecting all file_ids from the *files* table.
    a. SELECT file_id FROM files
2. For every file_id selected an external script update_file.php is called.
    a. php update_file.php file_id

update_file.php <file_id>

1. Retrieves all IPs which are currently seeding the file_id
2. Using the *peers table* in the Master database the script calculates whether or not the file is present on enough seeds per zone as well as whether the file is present in enough zones.
    a. If the file is not present in enough zones.
        i. Script randomly chooses zones
        ii. Adds the file to zones.
    b. If the file is not present on enough seeds per zone.
        i. Script calculates how many seeds in the zone currently have the file.
        ii. Adds the appropriate amount of additional seeds within that zone.
3. If additional peers are added for the file using this script the script rss_create.php is called for each peer that is affected.
    a. php rss_create.php <IP>

rss_create.php <IP>

1. Using the IP the script queries the *<IP> table* from the Master database.
2. For every file_id that is returned an item is added to the rss.xml file.
3. The rss.xml file is written to /home/joey/public_html/rss/<IP>/rss.xml.

After completion of the refresh_all.php script, all peers affected will have RSS feeds generated within their directories. Peers query the RSS feed every 5 minutes and automatically download the files in their feed.

3.4.6 PHP Scripts Described in Detail

addfile.php

Usage: Visit http://192.168.0.46/~joey/addfile.php to use. (Figure 3.7)

Includes: db_connect.php, Torrent.php

Description: This script is utilized via web interface in order to add files that one wishes to upload to the system. The user can configure the file to upload, number of seeds per zone, and number of zones that the file should be present in.

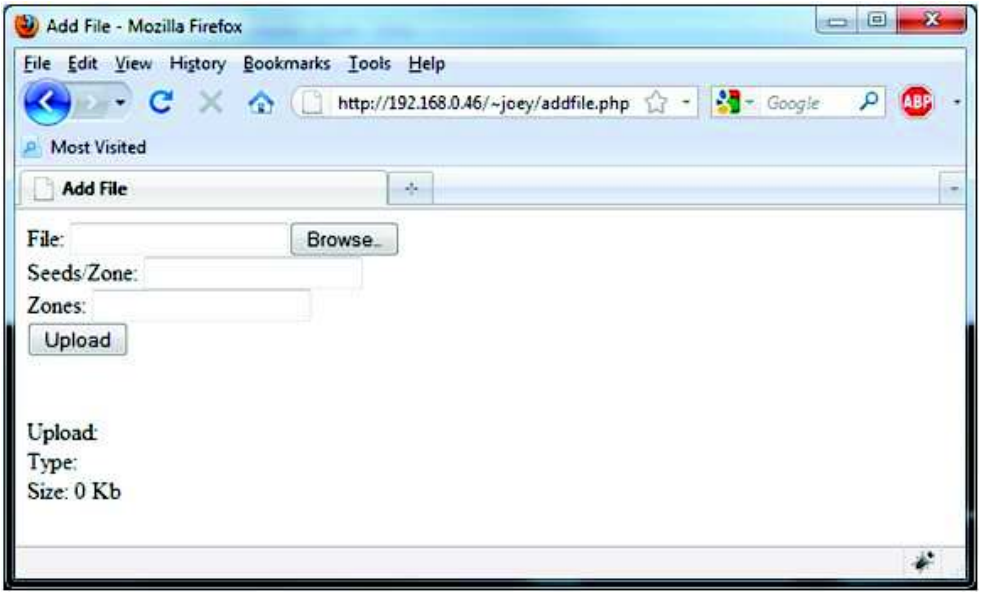


Figure 3.7: addfile.php script

Details:

1. The user visits the URL http://192.168.0.46/~joey/addfile.php
    a. Chooses file to upload.
    b. Sets the number of seeds per zone file should be present in.
    c. Sets the number of zones the file should be present in.
    d. Clicks upload.
2. The file is uploaded to the server using a POST request.
3. File is moved from temporary location to /home/joey/public_html/uploads/
4. A torrent file is created for the file which contains the metadata necessary for adding the file to the BitTorrent tracker. (See Torrent.php)
5. The file is added to the BitTorrent tracker using the torrent file created.
    a. This step executes an external script tracker_interface.php. (See tracker_interface.php)
6. A MySQL query is executed updating the *files* table in the Master database.
    a. INSERT into files (filename, file_id, seeds, zones)
7. Script returns success or failure.

addpeer.php

Usage: Visit http://192.168.0.46/~joey/addfile.php to use. (Figure 3.8)

Includes: db_connect.php

Description: This script is used in order to add peers to the *peers table* in the Master database. If there are currently peers in the database it will also list them at the bottom of the page. The IP is the IP of the peer, zone is the zone that the peer resides in, and space is the space in megabytes that the peer has available to store files.

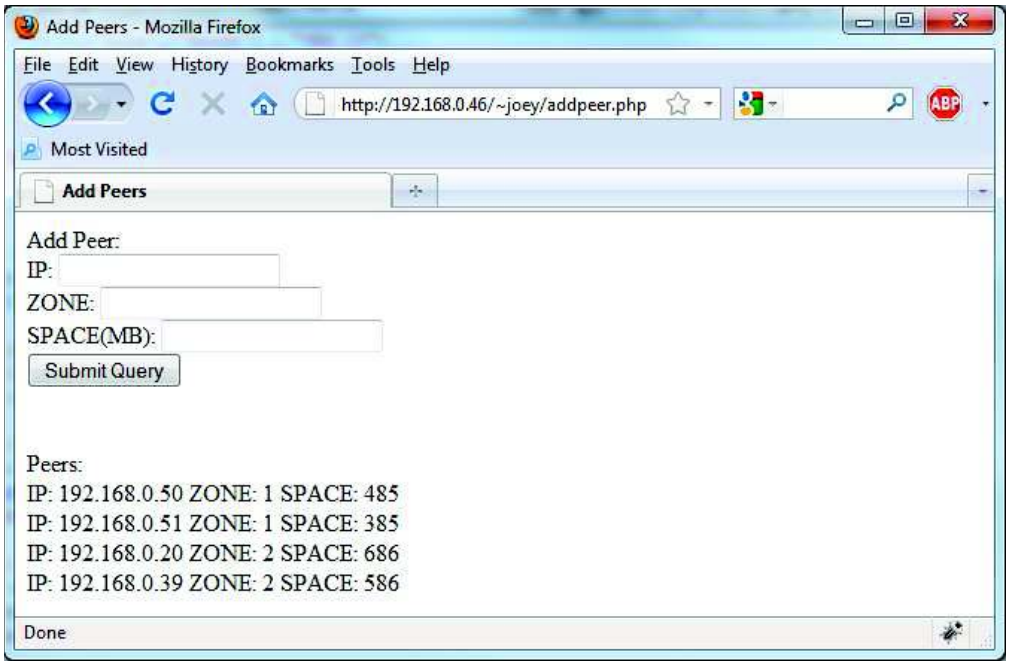


Figure 3.8: addpeer.php script

Details:

1. The user visits the URL http://192.168.0.46/~joey/addpeer.php and fills in the following:
    a. IP address of peer to add.
    b. Zone that peer resides in.
    c. Space in megabytes available for files.
2. Form is submitted using GET request to web server.
3. Peer is added into the *peers table* in the Master database.
    a. INSERT into peers (ip,zone,space)
4. Table is created in the Master database using the IP given for table name.
    a. *Note IP address is hyphenated due to MySQL not allowing for '.' in table names.*
5. RSS directory is created for the IP and permissions modified to 777.
    a. mkdir ./rss/$ip && chmod 777 ./rss/$ip
6. All peers are queried from *peers table* and listed on bottom of page.

db_connect.php

Usage: Included in other scripts, no direct usage.
Includes: None
Description: This file is included in many other scripts. It is used to make a connection to the database and select the Master database.

Details:

1. Creates a MySQL connection to the Master database.
2. Selects the Master database.
3. If it fails it will call die() ending all execution.

refresh_all.php

Usage: php refresh_all.php
Includes: db_connect.php
Description: Script called by cronjob every 15 minutes to refresh the status of all files that are maintained by the system.

Details:

1. MySQL query executed selecting all file_ids from the *files* table.
    a. SELECT file_id FROM files
2. For every file_id selected the script update_file.php is called (see update_file.php).
    a. php update_file.php $file_id

rss_create.php

Usage: php rss_create.php <IP>
Includes: None
Description: This script is used to create the rss.xml file for the given IP. It determines the contents of the rss.xml file based on the contents of the table *<IP>* in the Master database. For every file that is in the table it creates an entry in the rss.xml file.

Details:

1. For given IP select all file_ids and filenames from the *<IP> table* in Master database.
    a. SELECT file_id,filename FROM <IP>
2. Open the file ./rss/<IP>/ rss.xml for writing.
3. Create the beginning of the rss.xml file containing xml version, rss version, and last build date.
4. For each <file_id> that is selected in (1)
    a. Query the namemap table in Rivet database for filename and size of file.
        i. SELECT filename,size FROM namemap WHERE info_hash = <file_id>
    b. Create an RSS item using the information obtained from (4a).
5. Create end of RSS feed (ex: </rss>)
6. Write file and close filestream.

Torrent.php

Usage: Included in other scripts, no direct usage.

Includes: None

Description: This is an open source class written by Adrien Gibrat < adrien.gibrat@gmail.com>. This class is used in order to create .torrent files for use with the BitTorrent Tracker.

Details:

1. Create torrent file using the following syntax:
    a. $torrent = new Torrent('<file location>', '<announceURL>')
        i. File location is the location of the file you wish to create a torrent for.
        ii. AnnounceURL is the URL to the tracker announce interface.
2. Save torrent file using the following syntax:
    a. $torrent->save('<file location>')
        i. File location is the location you wish to save the file.
3. Retrieve hash_info (file_id) of torrent file:
    a. $torrent->hash_info()

tracker_interface.php

Usage: php tracker_interface.php <torrentLocation> <httpSeedLocation>

Includes: None

Description: This script is a helper script called from within addfile.php to interface with the tracker. Its main objectives are to authenticate with the tracker and add the file to the tracker via the web interface.

Details:

1. Creates cURL object in order to authenticate with tracker and store the authentication cookie in ./cookies/cookies.txt.
    a. The following cURL options were used:

cURL Option	Value	Purpose
CURLOPT_URL	http://192.168.0.46/~rivet/login.php	URL to execute.
CURLOPT_RETURNTRANSFER	1	Return output as a string.
CURLOPT_POST	TRUE	Perform a POST request
CURLOPT_COOKIEJAR	./cookies/cookies.txt	Location to save cookies.
CURLOPT_COOKIEFILE	./cookies/cookies.txt	Cookie file to use.
CURLOPT_FOLLOWLOCATION	1	Follow redirects.
CURLOPT_POSTFIELDS	'f_user' => 'joey', 'f_pass' => 'netsys', 'LogIn' => 'Log+In', 'legalterms' => 'on'	Fields to POST.

Table 3.12: cURL options used in tracker_interface.php

2. Executes login utilizing POST request on http://192.168.0.46/~rivet/login.php .
3. The following cURL options were changed:

cURL Option	Value	Purpose
CURLOPT_URL	http://192.168.0.46/~rivet/newtorrents.php	URL to execute.
CURLOPT_POSTFIELDS	torrent' => "@$torrentLoc", 'getrightseed' => 'enabled', 'httpftplocation' => "$httpseedLoc", 'autoset' => 'enabled'	Fields to POST.

Table 3.13: cURL options changed.

   a. The values for CURLOPT_POSTFIELDS represent the values needed in order to successfully add a torrent via the web interface. The torrent location is set, HTTP seeding is enabled, and the location of the HTTP seed is set.
4. The cURL object is executed again effectively adding the torrent to the tracker.

update_file.php

Usage: php update_file <file_id>

Includes: None

Description: This script is used to update a single file on the system. It ensures that the file exists on enough seeds per zone and in enough zones. If any of these are missing it then goes on to determine where to place seeds and adds them appropriately.

Details:

1. Given <file_id> retrieve all IPs from x<file_id> table in Rivet database. This is effectively retrieving all active seeders for the given file.
2. Calculate how many seeds per zone the file exists in and how many zones it exists in.
   a. For each IP query *peers table* in the Master database to determine which zone the IP exists in.
   b. Total number of zones, and seeds per zone.
3. Check to ensure the file is present in enough zones.
   a. If NOT:
      i. Determine the zones the file is not present in.
      ii. Randomly choose additional zones to add file to.
      iii. Query *files* table in Master database to determine how many seeds to add to additional zones
      iv. Call addPeer function for each seed to add. (See addPeer function below)
4. Check to ensure the file is on enough seeds per zone.
   a. If NOT:
      i. Determine number of seeds to add for current zone.

ii. Call addPeer function for each seed needed in zone. (See addPeer function below)

addPeer function – Adds number of peers, to specified zone, using fileID.
Usage: addPeer($numPeers, $zone, $fileID)

- int numPeers – Number of peers to add.
- int zone – zone peers should exist in.
- string fileID – file_id hash of file.

1. Retrieves the filename and size for <file_id> from *namemap* table in Rivet database.
    a. Converts size from bytes to megabytes and rounds up.
2. Selects IPs from *peers table* in Master database where the zone is the one specified in the call to addPeer. This selection is ordered by the amount of space left so that peers with the highest amount of space are selected first.
3. Each peer returned from (2) is checked to see whether or not the file already exists on that peer.
    a. IF NOT
        i. Insert the file into the *<IP> table* in Master database.
        ii. Update the *peers table* by decrementing the amount of space left for the peer.
        iii. Execute php rss_create.php <IP>.
    b. ELSE
        i. Goto the next available peer.
4. Check to see if enough peers have been added, if so end the loop.

3.5 Limitations

3.5.1 Security

The scope of this thesis does not include any security measures for the system. The following items are major security concerns in the current design:

Clients

All files that are downloaded by peers within the system are accessible by any person that uses the PC. This means that any file that is distributed by the system is readable by anyone. Additionally users could delete or modify the file on the PC.

RSS Feeds

The current design utilizes RSS feeds for each of the peers within the system. These RSS feeds are insecure and could be accessed by anyone who is able to access the URL http://192.168.0.46/~joey/rss/.

Torrent Files

The torrent files that are stored on the tracker are world readable. This means that anyone who is able to gain access to this via a web interface will be able to download and access the torrent files stored within the system.

3.5.2 File Size

The current configuration of the system only allows for a max files size of 1000 megabytes to be uploaded. This is due to the fact that the php.ini file is configured to only accept files which are 1000 megabytes or smaller.

The value was chosen due to the fact that POST requests are being used to upload the file to the server from a web browser. As the file size increases the chance of a timeout between the client and the server also increases. The size 1000 megabytes allowed for relatively large files to be uploaded while decreasing the chance of a timeout.

3.5.3 Deletion

Once a file is uploaded to the system there is currently no mechanism to delete the file. This is due to the fact that additional client software would be needed in order to instruct the clients to delete the file.

3.5.4 Authentication

The current authentication method used for the BitTorrent tracker only allows for a single user to be configured for access.

3.5.5 Front End

There are no front end interfaces or install scripts available for configuration of the system. All initial installation and configuration must be done manually on both the server side and client side.

4

Results

4.1 Testing Design

4.1.1 Testing Materials

Files:

- 900MB binary file (Disk Image)
- 900MB ASCII file (Generated ASCII text)

A 900MB file was used in each circumstance due to the speed of the transfer. If a smaller file had been used the transfer would have happened in a matter of seconds. Since a larger file took longer to transfer, it was easier to note important times such as when 50% or 99% of a file had been transferred.

Clients:

Three separate clients were used in order to facilitate testing. The same tests measuring RAM, CPU, and network utilization during file transfer were taken on all three separate clients. All clients were tested at separate times.

All clients had the following tests performed on them prior to testing:

- A hard drive disk check was run to ensure hard drives were properly functioning.
- A memory test was run to ensure there was no bad memory.
- A CPU test was performed in order to ensure no CPU malfunctions.

4.1.2 Measuring Memory Utilization

Tool: RamMap
Purpose: Measure RAM utilization for processes and files.

Summary of Usage:

1. Fresh Boot (+ 10 Minutes) – Measure system RAM utilization and service RAM utilization.
2. Initiation of file transfer
3. 50% completion of file transfer
4. 99% completion of file transfer
5. 100% completion of file transfer (+10 Minutes)

Description:
The tool RamMap is utilized in order to monitor each client system's RAM utilization. It has the ability to break down RAM utilization into granular activity such as by process and by file RAM utilization. This is advantageous due to the fact that torrent files may exist in memory before they are written to disk. Also, a torrent process will be running on each client machine which will also affect the RAM utilization of the machine. Additionally this tool also displays processes and files in RAM which are pre-cached for faster access using Windows Superfetch.

Usage:

1. Initial RAM measurements were taken of the client machine after the system was booted and finished pre-caching memory from the disk. In order to maintain consistent results the measurement was taken 10 minutes after Windows had loaded.

2. Another RAM measurement was taken at the initiation of a file transfer. This is after a client had realized it needed to download a file and the transfer had commenced. The uTorrent.exe RAM usage as well as total RAM usage was noted.

3. After 50% of the file had been transferred another measurement of both the uTorrent.exe RAM usage as well as the total RAM usage was measured.

4. At 99% of the file transfer a measurement was taken of the uTorrent.exe RAM usage and total RAM usage. This measurement is intended to take place just before the completion of the transfer in order to measure the potential 'maximum' usage of RAM.

5. A final RAM measurement was taken on the client machine 10 minutes after 100% of the file had been transferred. This measurement is intended to be used in order to compare the baseline measurement with the final effects of the uTorrent.exe on the machine.

4.1.3 Measuring CPU Utilization

Tool: Windows Performance Analyzer (Xperf)
Purpose: Measure CPU Utilization over given period.

Description:
Xperf is used in order to trace the performance of specific events that happen on a Windows system. This tool is used by turning on the monitoring at the start of the test period and stopping it when the period is over. After the data has been collected the tool is able to display the CPU utilization information over time. Additionally it can be used in order to see which processes over the period of time had the most CPU utilization in both milliseconds as well as a percentage of the total CPU power.

Usage:

The monitoring was configured to start 10 minutes after a fresh boot of the machine had occurred; this was to ensure that no measurements were affected by startup processes. Once started the following times were noted in order to compare the results to events of the service:

1. Initial time (Baseline)

2. Initiation of file transfer.
3. 50% completion of file transfer.
4. 99% completion of file transfer.
5. 100% completion of file transfer (+10 minutes)

4.1.4 Measuring Network Utilization

Tool: Hyperic HQ
Purpose: Measure individual client resources including: Network, CPU, Memory

Description:
Hyperic HQ is a powerful client server tool which is capable of monitoring multiple resources across many individual clients simultaneously. A program is used on each of the clients in order to monitor the network and resource usage of the machine. This client reports back to the server which then is able to compile the data into readable graphs. The most advantageous part of this tool is the ability to set time intervals for which to monitor data. After setting a time interval to view, the data is graphed appropriately allowing one to compare and contrast what was happening during this interval with the data that is gathered.

Usage:
Once the experiment had commenced the start time and completion time of transfers was noted. Additionally the following times were noted in order to be consistent with the previous experiments:

1. Initiation of file transfer.
2. 50% completion of file transfer.
3. 99% completion of file transfer.
4. 100% completion of file transfer (+10 minutes)

Post Test:
Network errors were noted after the completion of the tests. The following command was used in order to determine the amount of network errors that had occurred:

- netstat -e

4.2 Memory Utilization

Throughout the experiments the amount of RAM utilization by the uTorrent.exe process changed very little. Overall the consumption of RAM ranged between 4 and 10 megabytes for this process. The following examples show that RAM utilization is overall low and unchanged when utilizing the uTorrent client on a machine within the system.

4.2.1 Baseline

The total RAM usage for a client after fresh boot + 10 minutes was around 700 megabytes. Figure 4.1 shows a measurement of RAM usage on a client 10 minutes after a fresh boot. The total amount of RAM usage for this client is 717 megabytes of RAM.

The RAM utilization for the uTorrent.exe after fresh boot + 10 minutes was approximately 4 megabytes. Figure 4.2 shows the RAM measurement for the uTorrent.exe process. It can be seen that at this point the uTorrent.exe process has a very small RAM footprint.

Usage	Total	Active
Process Private	240,832 K	235,892 K
Mapped File	378,848 K	83,224 K
Shared Memory	63,888 K	28,544 K
Page Table	9,100 K	9,096 K
Paged Pool	84,308 K	82,856 K
Nonpaged Pool	190,556 K	190,548 K
System PTE	46,800 K	43,172 K
Session Private	19,352 K	19,352 K
Metafile	34,104 K	12,568 K
AWE		
Driver Locked	21,520 K	21,520 K
Kernel Stack	8,916 K	7,700 K
Unused	3,087,168 K	
Total	4,185,392 K	734,472 K

Figure 4.1: Total RAM usage after fresh boot + 10 minutes.

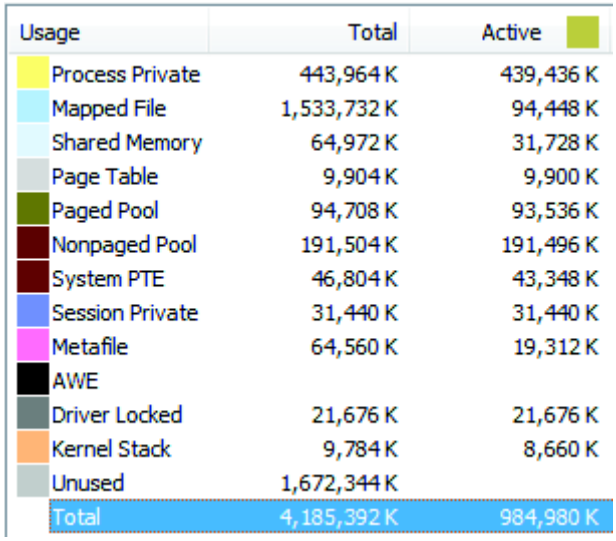
Process	Session	PID	Private
wrapper-win...	0	1448	1,096 K
WmiPrvSE.exe	0	1548	1,984 K
winlogon.exe	1	908	2,288 K
wininit.exe	0	428	1,192 K
uTorrent.exe	1	2368	4,068 K

Figure 4.2: uTorrent.exe RAM usage after fresh boot + 10 minutes.

4.2.2 Initiation of Transfer

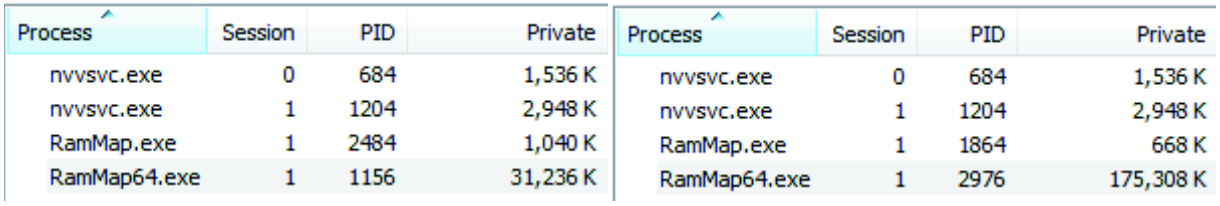
Another RAM measurement was taken at the initiation of the file transfer. Figure 4.3 shows the total amount of RAM usage for the same machine that was measured previously. Although the RAM usage appears to have increased greatly it is not as a result of the uTorrent.exe process. The total amount of RAM was affected by the RAM snapshots being taken by RamMap. Figure 4.4 shows the total amount of RAM used by RamMap greatly increases after a snapshot has been taken. This is due to the save file being cached in memory.

Figure 4.5 shows the RAM utilization of the uTorrent.exe client on the same machine that the previous measurements were taken. It can be seen that the amount of RAM utilized by uTorrent.exe has only increased to 7 megabytes from the previous 4 megabytes at baseline.



Usage	Total	Active
Process Private	443,964 K	439,436 K
Mapped File	1,533,732 K	94,448 K
Shared Memory	64,972 K	31,728 K
Page Table	9,904 K	9,900 K
Paged Pool	94,708 K	93,536 K
Nonpaged Pool	191,504 K	191,496 K
System PTE	46,804 K	43,348 K
Session Private	31,440 K	31,440 K
Metafile	64,560 K	19,312 K
AWE		
Driver Locked	21,676 K	21,676 K
Kernel Stack	9,784 K	8,660 K
Unused	1,672,344 K	
Total	4,185,392 K	984,980 K

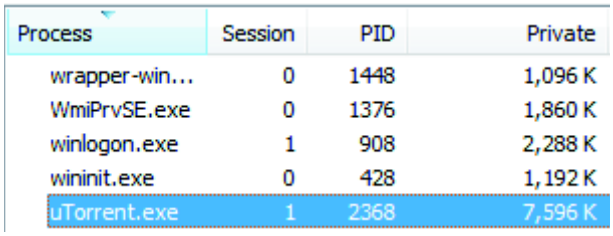
Figure 4.3: Total amount of RAM usage after initiation of the file transfer.



Process	Session	PID	Private
nvvsvc.exe	0	684	1,536 K
nvvsvc.exe	1	1204	2,948 K
RamMap.exe	1	2484	1,040 K
RamMap64.exe	1	1156	31,236 K

Process	Session	PID	Private
nvvsvc.exe	0	684	1,536 K
nvvsvc.exe	1	1204	2,948 K
RamMap.exe	1	1864	668 K
RamMap64.exe	1	2976	175,308 K

Figure 4.4: RamMap RAM utilization increasing as a result of taking RAM snapshots.



Process	Session	PID	Private
wrapper-win...	0	1448	1,096 K
WmiPrvSE.exe	0	1376	1,860 K
winlogon.exe	1	908	2,288 K
wininit.exe	0	428	1,192 K
uTorrent.exe	1	2368	7,596 K

Figure 4.5: uTorrent.exe RAM usage at initiation of the file transfer.

4.2.3 Transfer at 50%

After 50% of a file had been transferred another measurement was taken. It can be seen in figure 4.6 at this point the total RAM utilization from the initial file transfer has only increased slightly.

Figure 4.7 shows the RAM utilization for the uTorrent.exe process at approximately 8 megabytes. This is a slight increase from the 7 megabytes measured at the initiation of a file transfer.

Usage	Total	Active
Process Private	449,912 K	445,248 K
Mapped File	1,584,644 K	95,048 K
Shared Memory	64,972 K	31,788 K
Page Table	9,972 K	9,968 K
Paged Pool	94,712 K	93,540 K
Nonpaged Pool	191,564 K	191,556 K
System PTE	46,804 K	43,348 K
Session Private	30,360 K	30,360 K
Metafile	65,332 K	19,316 K
AWE		
Driver Locked	21,676 K	21,676 K
Kernel Stack	9,548 K	8,400 K
Unused	1,615,896 K	
Total	4,185,392 K	990,248 K

Figure 4.6: Total amount of RAM usage at 50% of the file transfer.

Process	Session	PID	Private
wrapper-win...	0	1448	1,096 K
WmiPrvSE.exe	0	1376	1,964 K
winlogon.exe	1	908	2,288 K
wininit.exe	0	428	1,192 K
uTorrent.exe	1	2368	8,524 K

Figure 4.7: uTorrent.exe RAM usage at 50% of the file transfer.

4.2.4 Transfer at 99%

Another RAM measurement was taken at 99% completion of the file transfer. It can be seen in figure 4.8 that again the total amount of RAM usage is almost unchanged.

Figure 4.9 is the measurement of the uTorrent.exe process at 99% of the file transfer. It can be seen that the RAM utilization has decreased from 8 megabytes (50% of transfer) to approximately 7 megabytes. The change in RAM for the uTorrent.exe process again is small compared to previous measurements.

Usage	Total	Active
Process Private	451,612 K	446,980 K
Mapped File	1,635,568 K	95,320 K
Shared Memory	65,068 K	31,888 K
Page Table	10,160 K	10,156 K
Paged Pool	94,820 K	93,648 K
Nonpaged Pool	191,696 K	191,688 K
System PTE	46,804 K	43,348 K
Session Private	30,296 K	30,296 K
Metafile	65,592 K	19,580 K
AWE		
Driver Locked	21,676 K	21,676 K
Kernel Stack	9,272 K	8,024 K
Unused	1,562,828 K	
Total	4,185,392 K	992,604 K

Figure 4.8: Total RAM usage at 99% of the file transfer.

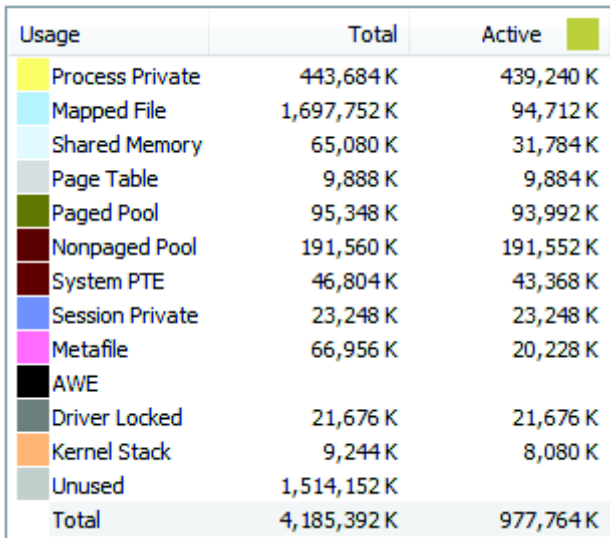
Process	Session	PID	Private
wrapper-win...	0	1448	1,096 K
WmiPrvSE.exe	0	1376	1,948 K
winlogon.exe	1	908	2,288 K
wininit.exe	0	428	1,192 K
uTorrent.exe	1	2368	6,864 K

Figure 4.9: uTorrent.exe RAM usage at 99% of the file transfer.

4.2.5 Final Measurement

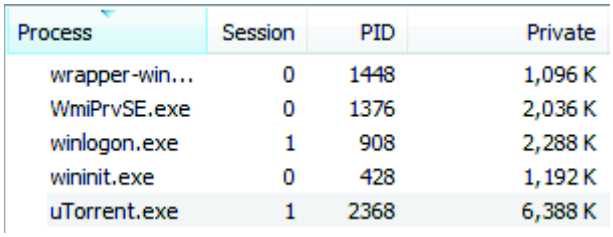
A final RAM measurement was taken 10 minutes after completion of the file transfer. Figure 4.10 shows the total amount of RAM usage on the machine at this point. It can be seen that compared to baseline (Figure 4.1) the total amount of RAM usage has increased by approximately 200 megabytes. This is mainly due to the fact that the RAM usage of RamMap has increased as a result of taking RAM snapshots as described above in the initiation of the file transfer.

A better representation of how the uTorrent.exe process has affected the system can be seen in figure 4.11 which shows the amount of RAM that the process uses 10 minutes after completion of the file transfer. Compared to the baseline measurement of approximately 4 megabytes the final measurement is only 6 megabytes.



Usage	Total	Active
Process Private	443,684 K	439,240 K
Mapped File	1,697,752 K	94,712 K
Shared Memory	65,080 K	31,784 K
Page Table	9,888 K	9,884 K
Paged Pool	95,348 K	93,992 K
Nonpaged Pool	191,560 K	191,552 K
System PTE	46,804 K	43,368 K
Session Private	23,248 K	23,248 K
Metafile	66,956 K	20,228 K
AWE		
Driver Locked	21,676 K	21,676 K
Kernel Stack	9,244 K	8,080 K
Unused	1,514,152 K	
Total	4,185,392 K	977,764 K

Figure 4.10: Total RAM usage 10 minutes after completion of the file transfer.



Process	Session	PID	Private
wrapper-win...	0	1448	1,096 K
WmiPrvSE.exe	0	1376	2,036 K
winlogon.exe	1	908	2,288 K
wininit.exe	0	428	1,192 K
uTorrent.exe	1	2368	6,388 K

Figure 4.11: uTorrent.exe RAM usage 10 minutes after completion of file transfer.

4.2.6 RAM Utilization over Time

Table 4.1 shows the raw data for the RAM utilization for each of the clients and tests run. Figure 4.12 is a graph of the values in table 4.1 and shows how RAM usage of uTorrent.exe changes over time.

Time	PC1, Binary (MB)	PC1, ASCII (MB)	PC2, Binary (MB)	PC2, ASCII (MB)	PC3, Binary (MB)	PC3, ASCII (MB)
Baseline	3.97	3.97	4.05	4.05	2.41	2.41
Initital	7.41	6.47	6.88	7.29	6.54	4.13
50% Transferred	8.32	8.34	8.48	7.96	8.01	5.72
99% Transferred	6.70	8.00	7.27	6.65	6.53	5.57
100% Transferred + 10 Minutes	6.24	6.79	6.89	6.61	6.09	4.07

Table 4.1: RAM Utilization values for all tests performed.

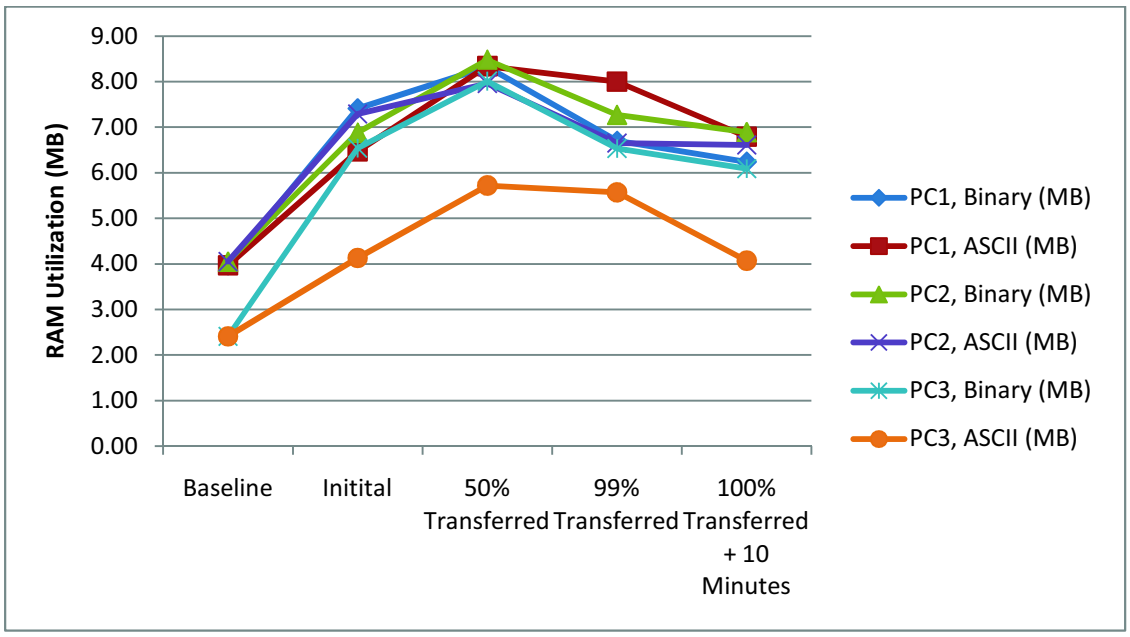


Figure 4.12: Graph of RAM utilization for all tests performed. Based on values from table 4.1.

4.2.7 ASCII vs. Binary Files

The type of file transferred had no effect on the amount of RAM utilized by the uTorrent.exe process. Figure 4.13 is a comparison of files transfers on the same client. On the left is the RAM usage for the uTorrent.exe process while transferring an ASCII file. The right depicts the RAM usage when transferring a binary file. It can be seen that the RAM usage is nearly identical.

Process	Session	PID	Private
wrapper-win...	0	1400	1,096 K
WmiPrvSE.exe	0	2996	1,932 K
winlogon.exe	1	920	2,268 K
wininit.exe	0	428	1,156 K
uTorrent.exe	1	2344	8,544 K

Process	Session	PID	Private
wrapper-win...	0	1448	1,096 K
WmiPrvSE.exe	0	1376	1,964 K
winlogon.exe	1	908	2,288 K
wininit.exe	0	428	1,192 K
uTorrent.exe	1	2368	8,524 K

Figure 4.13: uTorrent.exe RAM usage at 50% of file transfer ASCII (Left) vs Binary (Right).

4.2.8 Other Test Results for RAM Utilization

The testing of all three clients produced similar results to those shown above. While it is normal for the total amount of RAM to shift slightly between test to test there were no machines which demonstrated any notable differences. Overall the total RAM usage as well as the uTorrent.exe RAM usage was very consistent from client to client.

4.3 CPU Utilization

During testing the CPU utilization was consistently between 5% and 10% load while a transfer was taking place. While idle, the uTorrent.exe process consumed almost no CPU load. First a baseline will be shown in order to compare an idle uTorrent.exe process to one during transfer. Next, examples will be shown of transfers taking place and the load that uTorrent.exe has on the CPU. Finally a final measurement will be shown which takes place 10 minutes after completion of a transfer.

4.3.1 Baseline

The baseline CPU load for uTorrent.exe is less than 0.2% of the total CPU. Figure 4.14 depicts the uTorrent.exe processor load as a percentage of the total CPU power. It can be seen that this process takes nearly no CPU load when it is idle.

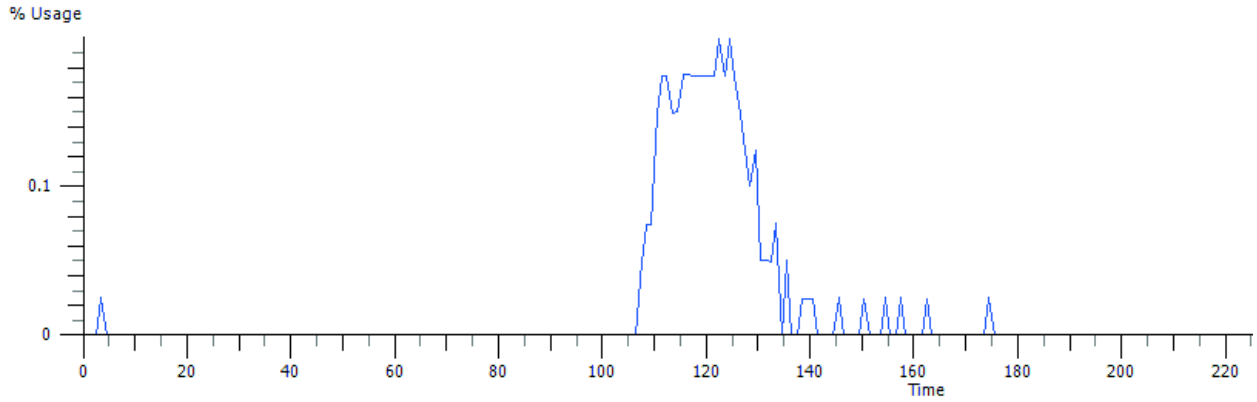


Figure 4.14: uTorrent.exe baseline CPU load.

4.3.2 CPU Load During Transfers

Each of the file transfers was measured for CPU load. During the file transfer the following times were noted in relation to the CPU measurement: initiation of the file transfer, 50% of the transfer, and 99% of the transfer. The following are examples of the times and how they relate to the CPU load for the uTorrent.exe process.

Transfer 1:

Table 4.2 describes the important times that were noted during the file transfer. This table can be referenced when looking at figure 4.15 which shows the CPU load during the transfer. The times in this table refer to the times on the x-axis for figure 4.15. It can be seen that once the transfer is initiated the average CPU load ranges between 5%-8%. After the transfer has finished the CPU load decreases dramatically.

% of File Transfer	Time(sec)
Initiation of Transfer	0
50% of Transfer	45
99% of Transfer	82

Table 4.2: Time events in relation to CPU load measurements for figure 4.15.

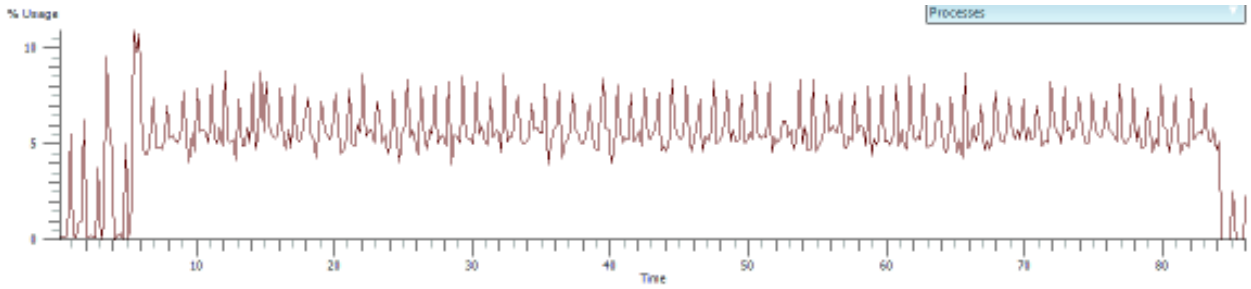


Figure 4.15: uTorrent.exe CPU load during transfer.

Transfer 2:

Table 4.3 describes the important times that occurred in relation to the CPU measurement in figure 4.16. The times in this table refer to the times on the x-axis for figure 4.16. This transfer was similar to transfer 1 but the CPU utilization was slightly higher, averaging between 6% -10% CPU load. Overall the same sequence of events occurs during transfer. The CPU is under load during transfer then drops off once transfer has completed.

% of File Transfer	Time(sec)
Initiation of Transfer	2
50% of Transfer	43
99% of Transfer	80

Table 4.3: Time events in relation to CPU load measurements for figure 4.16.

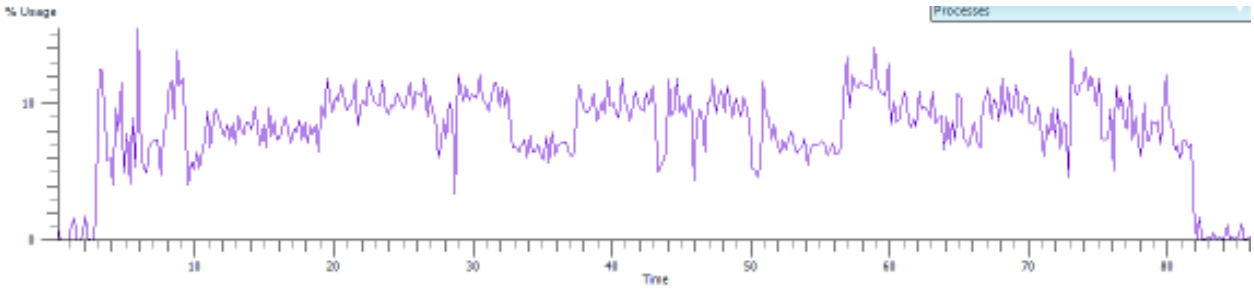


Figure 4.16: uTorrent.exe CPU load during transfer.

4.3.3 Final Measurement

A final measurement was taken of the CPU load 10 minutes after the completion of a file transfer. Figure 4.17 shows the CPU load of the uTorrent.exe process. It can be seen that there is a noticeable increase of 1% CPU load after a transfer has taken place. This may be due to the fact that the client will now notify the BitTorrent tracker that it has the file, as well as keep track of the torrent information.

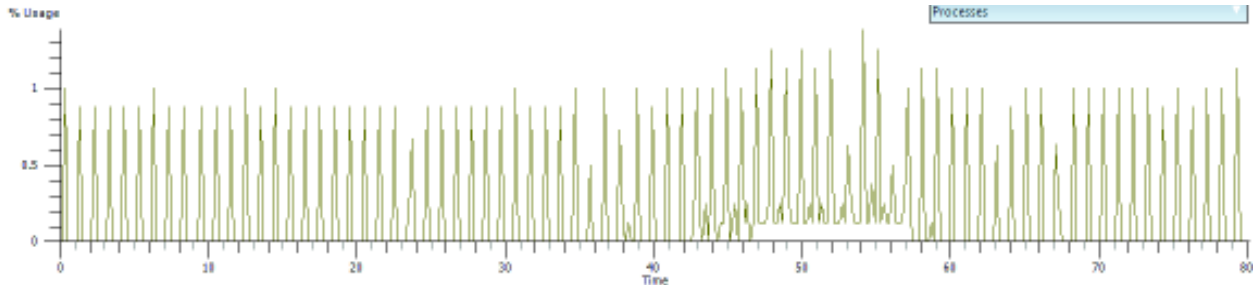


Figure 4.17: uTorrent.exe CPU load 10 minutes after transfer.

4.3.4 ASCII vs. Binary File

There were no notable differences when transferring a binary file versus an ASCII file. Figure 4.18 depicts the uTorrent.exe processor load while transferring a Binary file. It can be seen that the CPU load averages between 5%-10%. Figure 4.19 depicts the uTorrent.exe processor load while transferring an ASCII file. The average load of the CPU is also between 5%-10% for the duration of the transfer.

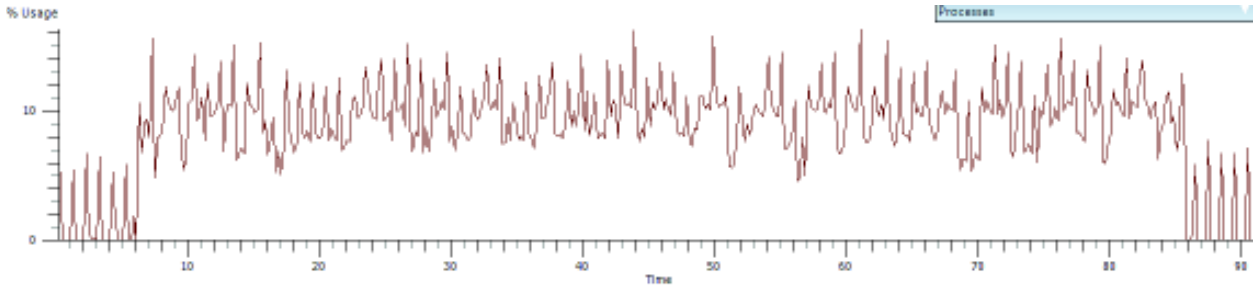


Figure 4.18: uTorrent.exe CPU load for Binary file transfer.

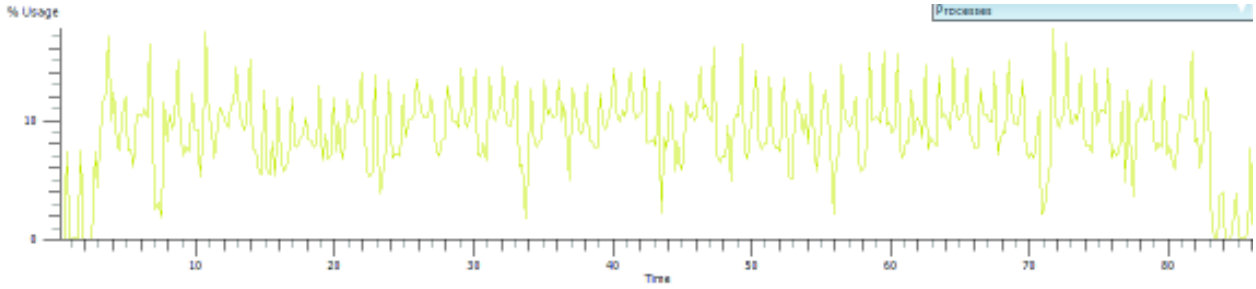


Figure 4.19: uTorrent.exe CPU load for ASCII file transfer.

4.3.5 Other Test Results for CPU Utilization

Throughout all of the tests run for the uTorrent.exe affect on CPU load there were no notable differences between tests. On average clients consumed between 5%-10% CPU load, varying little from test to test. After the completion of a transfer the client's CPU load increased by approximately 1% from baseline as a result of downloading a file. Overall results were consistent between all three separate clients tested.

4.4 Network Utilization

While testing the network utilization was between 70Mbps – 90Mbps download and 1300Kbps – 1800Kbps upload during a file transfer. Before testing, the idle network traffic was measured on the clients. This data is used in order to show that clients use very little bandwidth while a file transfer is not taking place.

4.4.1 Idle Network

While idle there is very little network traffic occurring. Figure 4.20 shows that over an 8 hour period there was on average 320bps of received network traffic for this client. Figure 4.21 shows for the same 8 hour period on average there was 229bps of network traffic transmitted for the same client. Overall the idle network traffic is very insignificant considering the environment is a 100Mbps network.

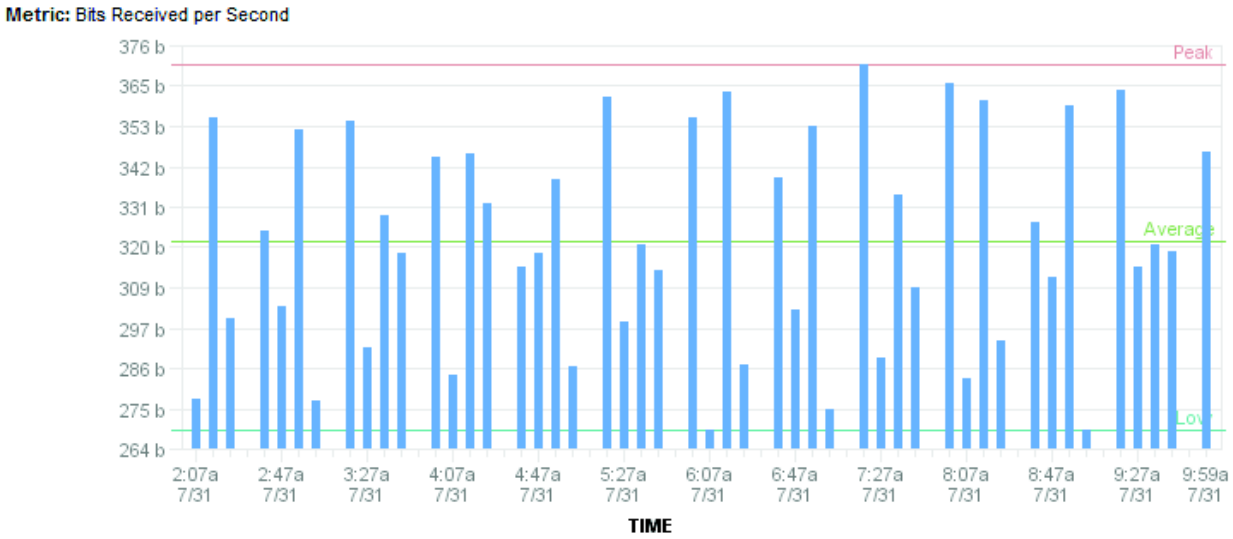


Figure 4.20: Bits receives per second over 8 hour period.

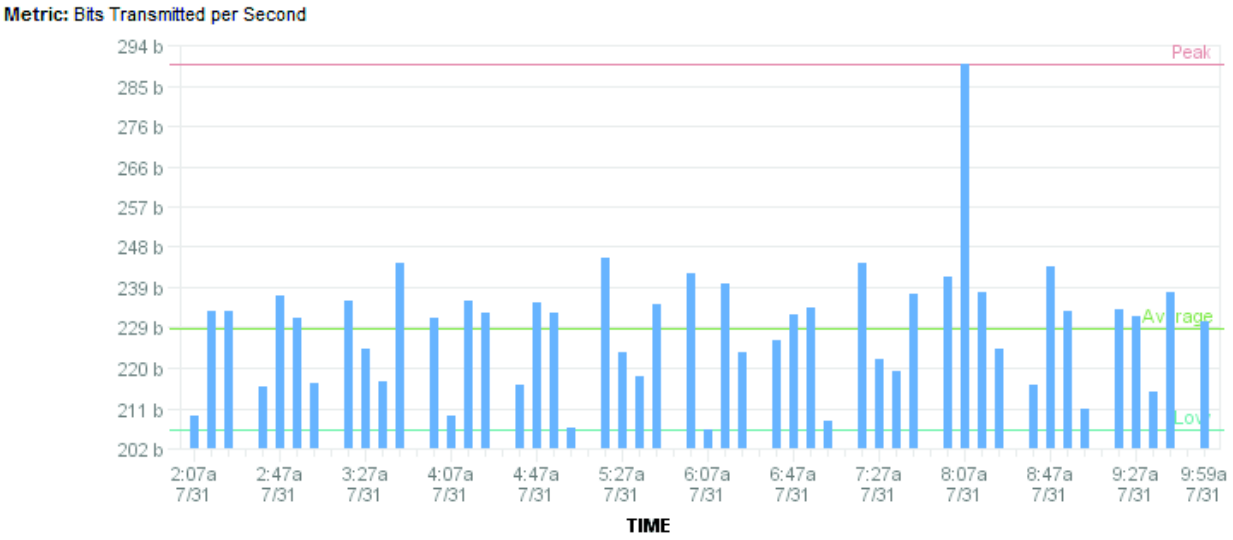


Figure 4.21: Bits Transmitted per second over 8 hour period.

4.4.2 Network Utilization Before, During, & After Transfers

The following transfers are examples of the network traffic measured during a BitTorrent file transfer within the system. Each of the clients measured during these tests were the only clients receiving the file at the time of download.

Transfer 1

During the file transfer the 100Mb downlink is almost fully saturated. Figure 4.22 shows a file transfer taking place and the downlink speed that is used. It can be seen that when the file transfer is started ~4:10p that the downlink is almost fully saturated at 69Mbps. After completion of the transfer ~4:14p the bandwidth used returns to almost nothing.

The uplink bandwidth for the same file transfer can be seen in figure 4.23. When the transfer starts the uplink quickly increases to ~1100Kbps. After the transfer has completed it returns to almost nothing.

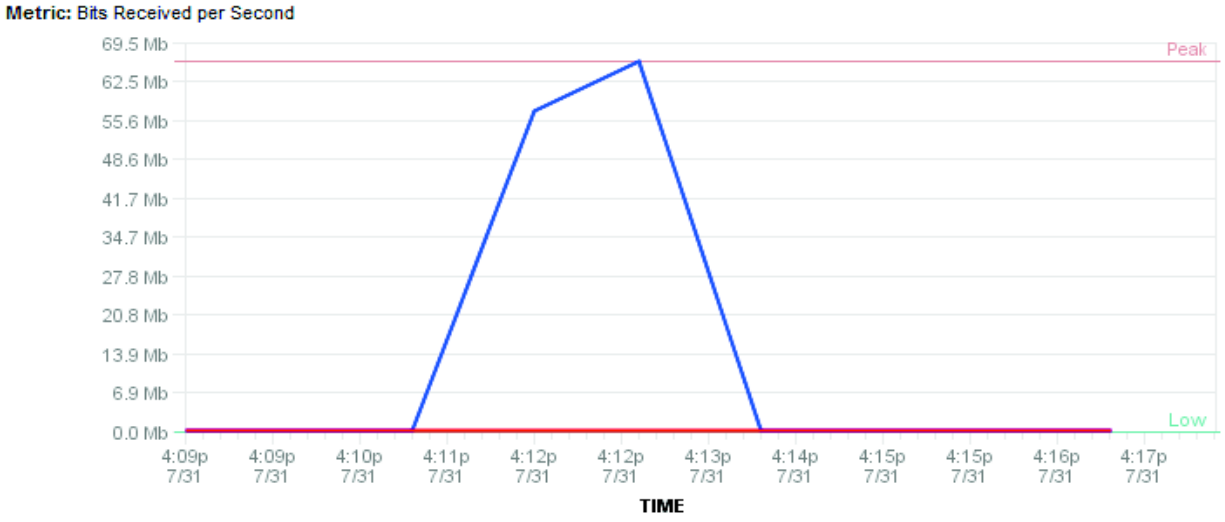


Figure 4.22: Bits recieved per second during file transfer.

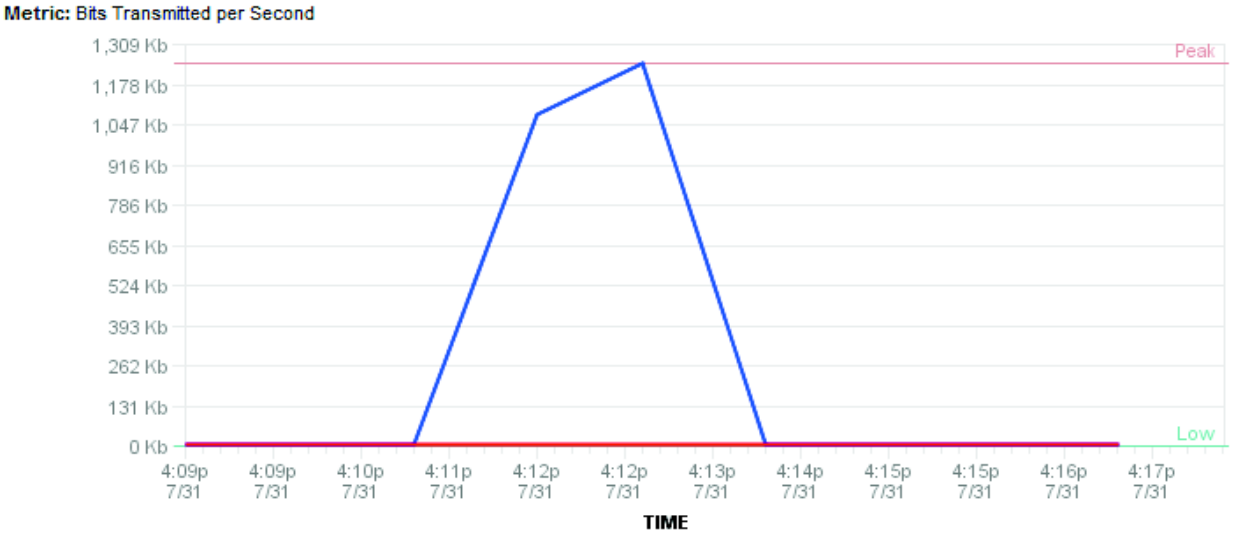


Figure 4.23: Bits transmitted per second during file transfer.

Transfer 2:

A second example of network traffic during a transfer also shows similar results. It can be seen in figure 4.24 when the transfer begins ~4:59p the downlink bandwidth consumed is 77Mbps. After completion of the transfer ~5:01p the bandwidth quickly decreases.

Figure 4.25 shows the uplink bandwidth consumed by the client during the same transfer. It can be seen that ~1500Kbps of uplink bandwidth is being consumed while the transfer takes place. After the file has completed the uplink decreases to almost nothing.

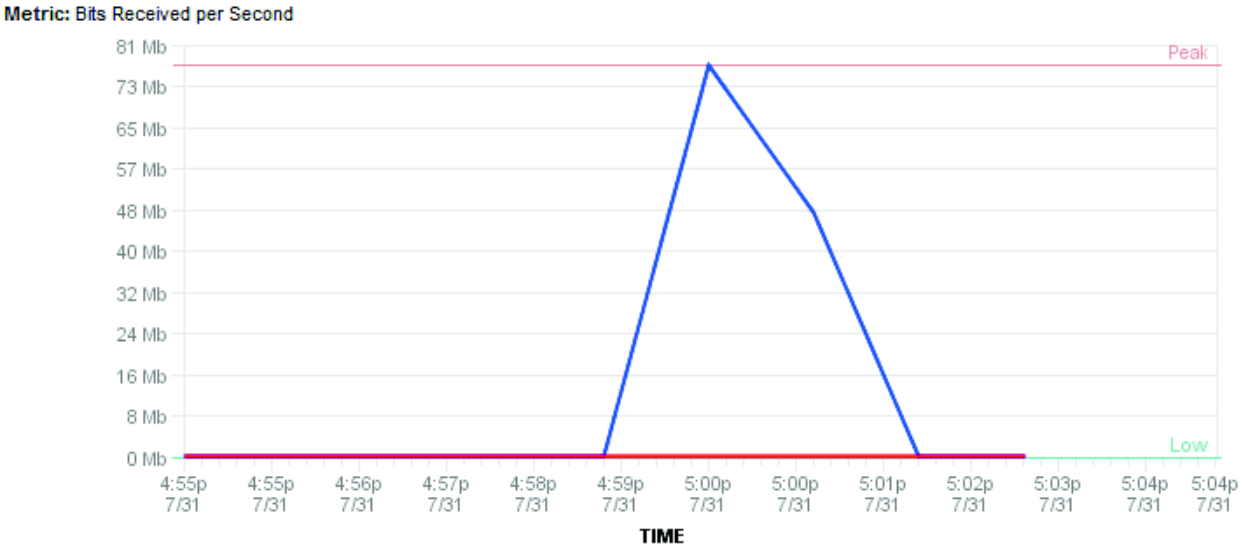


Figure 4.24: Bits received per second during file transfer.

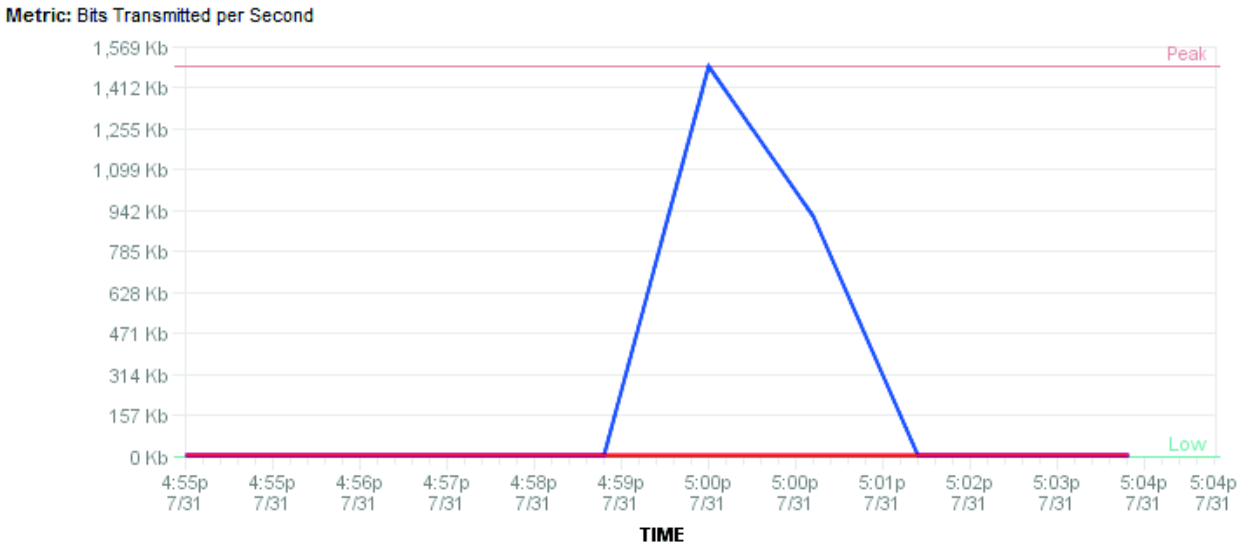


Figure 4.25: Bits transmitted per second during file transfer.

4.4.3 ASCII vs. Binary

The following figures demonstrate that there are no differences in network utilization when transferring an ASCII file as opposed to a binary file. Figure 4.26 demonstrates the downlink bandwidth utilized during an ASCII file transfer. The bandwidth quickly increases once initiated, the peak transfer rate is 90Mbps, and the bandwidth quickly decreases after the file has finished. Figure 4.27 shows the uplink utilized for the same file and it follows the same behavior except its peak is ~1800Kbps.

Figures 4.28 and 4.29 show the same behaviors and speed as figures 4.26 and 4.27. The file being transferred in figures 4.28 and 4.29 is a binary file rather than an ASCII file, but shows no difference in performance or utilization.

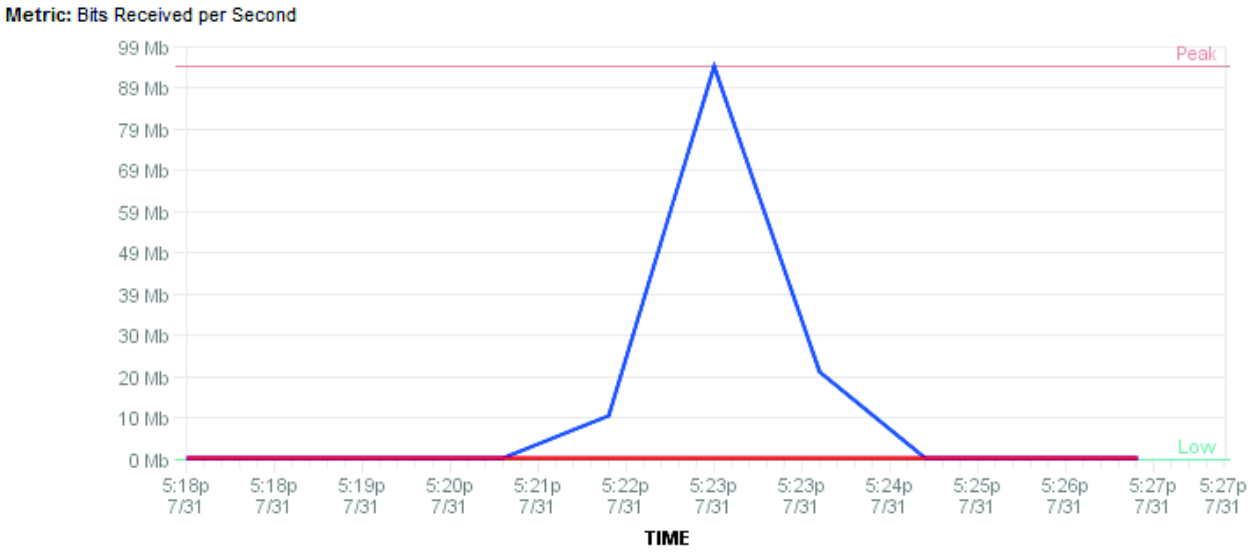


Figure 4.26: Bits received per second during file transfer. (ASCII File)

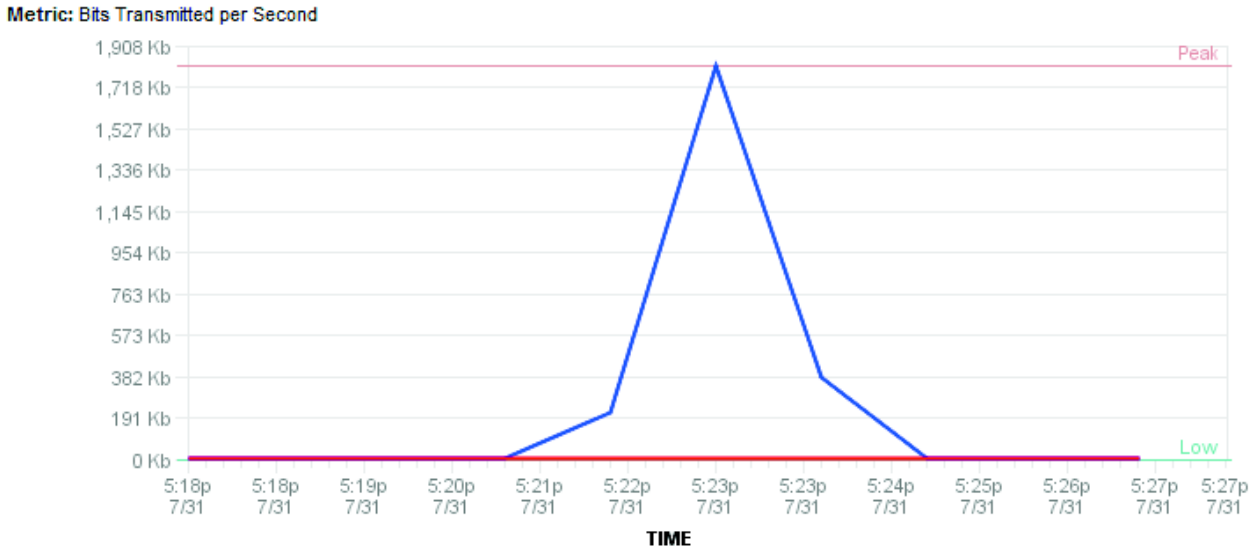


Figure 4.27: Bits transmitted per second during file transfer. (ASCII File)

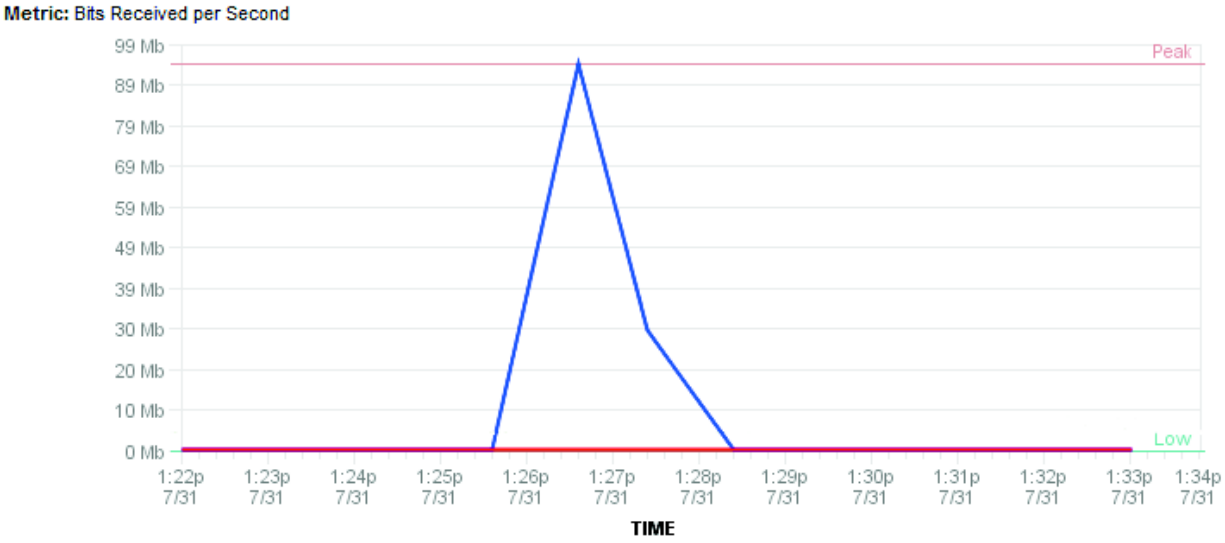


Figure 4.28: Bits received per second during file transfer. (Binary File)

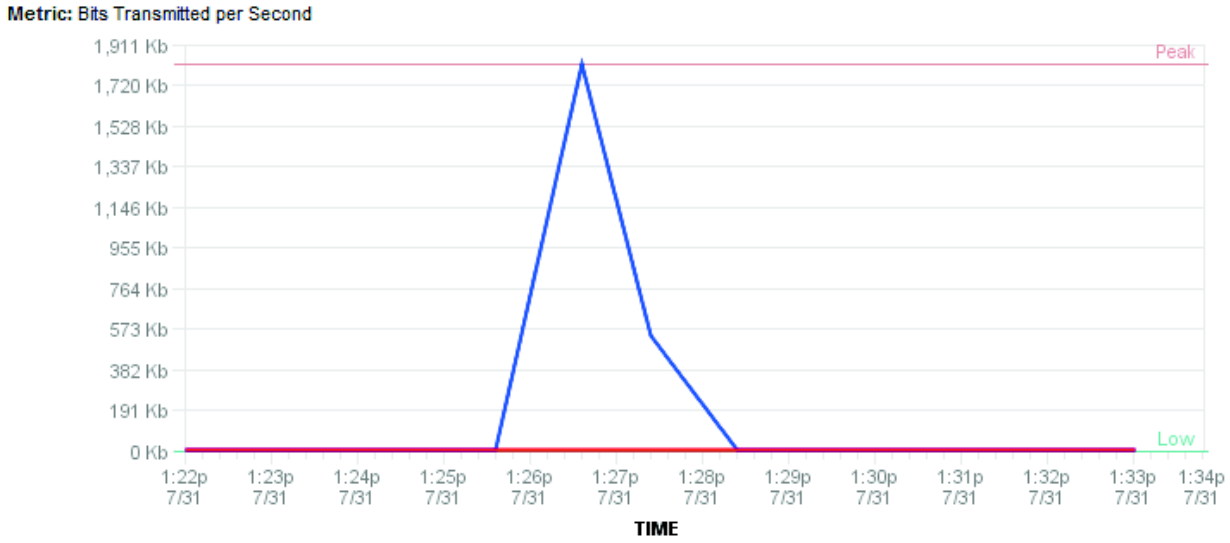


Figure 4.29: Bits received per second during file transfer. (Binary File)

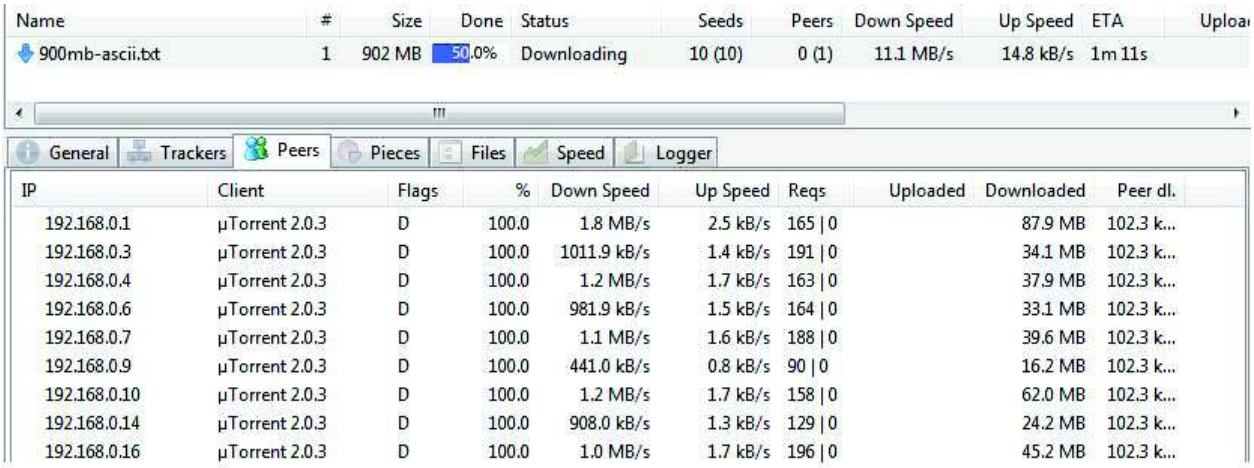
4.4.4 Other Test Results for Network Utilization

There were small variations between the max downlink and uplink speeds from test to test. These ranged from 70Mbps – 90Mbps peak speed for downlink and 1300Kbps – 1800Kbps for uplink speed. Other than these speed differences between tests the overall behavior and general speeds were consistent between each of the three clients that were tested.

4.5 Validation of Protocol

When a client downloads a file from the BitTorrent tracker it receives a list of available seeds from which it can download the file from. Figure 4.30 shows an example of the list of peers that the client 192.168.0.8 has received for the file 900mb-ascii.txt. One can see at the bottom of figure 4.30 that there are many peers that the client has connected to and is downloading the file from.

Figure 4.31 validates that the client is indeed downloading the file from these multiple peers simultaneously. In the Wireshark packet capture the peers 192.168.0.1, 192.168.0.4, and 192.168.0.14 are highlighted. They are utilizing the BitTorrent protocol in order to transfer the file 900mb-ascii.txt to the destination 192.168.0.8. These clients also appear in figure 4.30 where the program uTorrent states that 192.168.0.8 is downloading from these peers as well as others.

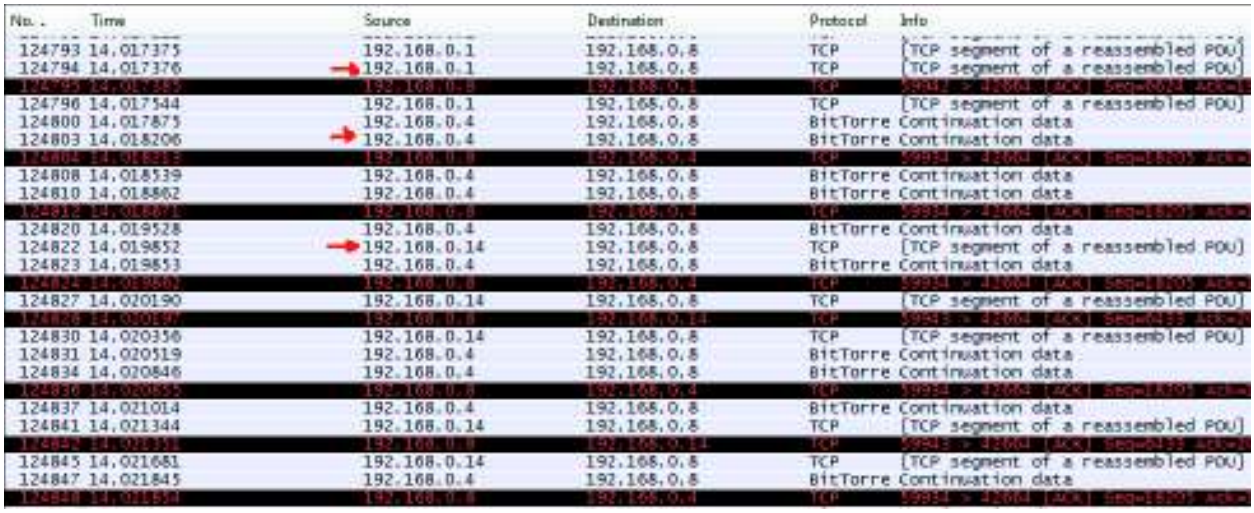


Name	#	Size	Done	Status	Seeds	Peers	Down Speed	Up Speed	ETA	Uploa
900mb-ascii.txt	1	902 MB	50.0%	Downloading	10 (10)	0 (1)	11.1 MB/s	14.8 kB/s	1m 11s	

General | Trackers | Peers | Pieces | Files | Speed | Logger

IP	Client	Flags	%	Down Speed	Up Speed	Reqs	Uploaded	Downloaded	Peer dl.
192.168.0.1	µTorrent 2.0.3	D	100.0	1.8 MB/s	2.5 kB/s	165 | 0		87.9 MB	102.3 k...
192.168.0.3	µTorrent 2.0.3	D	100.0	1011.9 kB/s	1.4 kB/s	191 | 0		34.1 MB	102.3 k...
192.168.0.4	µTorrent 2.0.3	D	100.0	1.2 MB/s	1.7 kB/s	163 | 0		37.9 MB	102.3 k...
192.168.0.6	µTorrent 2.0.3	D	100.0	981.9 kB/s	1.5 kB/s	164 | 0		33.1 MB	102.3 k...
192.168.0.7	µTorrent 2.0.3	D	100.0	1.1 MB/s	1.6 kB/s	188 | 0		39.6 MB	102.3 k...
192.168.0.9	µTorrent 2.0.3	D	100.0	441.0 kB/s	0.8 kB/s	90 | 0		16.2 MB	102.3 k...
192.168.0.10	µTorrent 2.0.3	D	100.0	1.2 MB/s	1.7 kB/s	158 | 0		62.0 MB	102.3 k...
192.168.0.14	µTorrent 2.0.3	D	100.0	908.0 kB/s	1.3 kB/s	129 | 0		24.2 MB	102.3 k...
192.168.0.16	µTorrent 2.0.3	D	100.0	1.0 MB/s	1.7 kB/s	196 | 0		45.2 MB	102.3 k...

Figure 4.30: BitTorrent Transfer Example



No.	Time	Source	Destination	Protocol	Info
124793	14.017375	192.168.0.1	192.168.0.8	TCP	[TCP segment of a reassembled PDU]
124794	14.017376	192.168.0.1	192.168.0.8	TCP	[TCP segment of a reassembled PDU]
124796	14.017544	192.168.0.1	192.168.0.8	TCP	[TCP segment of a reassembled PDU]
124800	14.017875	192.168.0.4	192.168.0.8	BitTorre	Continuation data
124803	14.018206	192.168.0.4	192.168.0.8	BitTorre	Continuation data
124808	14.018539	192.168.0.4	192.168.0.8	BitTorre	Continuation data
124810	14.018862	192.168.0.4	192.168.0.8	BitTorre	Continuation data
124820	14.019528	192.168.0.4	192.168.0.8	BitTorre	Continuation data
124822	14.019852	192.168.0.14	192.168.0.8	TCP	[TCP segment of a reassembled PDU]
124823	14.019853	192.168.0.4	192.168.0.8	BitTorre	Continuation data
124827	14.020190	192.168.0.14	192.168.0.8	TCP	[TCP segment of a reassembled PDU]
124830	14.020356	192.168.0.14	192.168.0.8	TCP	[TCP segment of a reassembled PDU]
124831	14.020519	192.168.0.4	192.168.0.8	BitTorre	Continuation data
124834	14.020846	192.168.0.4	192.168.0.8	BitTorre	Continuation data
124837	14.021014	192.168.0.4	192.168.0.8	BitTorre	Continuation data
124841	14.021344	192.168.0.14	192.168.0.8	TCP	[TCP segment of a reassembled PDU]
124845	14.021681	192.168.0.14	192.168.0.8	TCP	[TCP segment of a reassembled PDU]
124847	14.021845	192.168.0.4	192.168.0.8	BitTorre	Continuation data

Figure 4.31: Packet Capture of BitTorrent Transfer Example

5

Conclusion

The proof of concept implementation was successful in that it was able to replicate files to multiple peers in different zones. It has shown that it is possible to utilize the BitTorrent technology in order to utilize unused hard disk space within the context of the problem. This means it is able to determine where a file is in relation to a computer lab as well as whether or not enough computers are hosting a file within a lab. From this the system is able to automatically determine whether or not additional hosts are needed for a file, as well as distribute the file to appropriate hosts.

The results show that the RAM utilization for the hosts within the system is relatively low for the uTorrent.exe process itself. The total amount of RAM utilized on the host is also relatively low, showing that this system has little effect on the RAM for the system. Even after a transfer had completed, the machine returned to near baseline measurements.

Similarly the CPU utilization for hosts within the system was relatively low. The maximum percentage of CPU utilization during transfer was only 10% of the host's full potential. One side effect of a completed transfer was that the host's CPU utilization did not fully return to baseline. It was shown that the CPU utilization after completion of a transfer was approximately 1% higher than that of baseline.

Network utilization was the highest used resource out of all that were measured. The hosts were utilizing between 70Mbps – 90Mbps for the downlink. After taking into consideration protocol overhead it can be determined that the network was the bottleneck in the entire system. Had this system been implemented on a 1Gbps network the RAM and CPU usage may have been much different.

As a result of utilizing the entire downlink bandwidth available for the host it is important to note that this would most likely affect the user experience on the machine. Limitations on network utilization on the client end may be needed in order to ensure that user experience is not affected when a transfer is taking place.

Overall the proof of concept system has proven that it is possible to create a modified BitTorrent system which allows for files to be distributed to multiple computers and multiple computing labs. The effect of the system on clients is low, with the exception of network utilization. Future work is needed in order to secure the system, as well as add additional features which would allow it to be used in real world environments.

6

Future Work

6.1 Functionality

There are many features that could be added on to the system in order to give it increased functionality. The following were huge limitations noted in the current implementation and could be addressed in future work on this project.

6.1.1 Deletion

One of the biggest features missing in a system such as this is there is no deletion mechanism. Currently, if a user wishes to delete a file from the system he or she must do so by hand. This includes removing database entries, as well as deletion of the actual data on the client machine. In order to implement this there must be some mechanism implemented on the client side that allows for deletion of files.

Additionally server side changes would be needed in order to allow for the removal of torrent files, as well as update the appropriate BitTorrent tracker databases.

6.1.2 Increased File Size

One limitation of the system is that file size is limited to 1000MB files. This is due to the fact that frequent timeouts were occurring when trying to upload larger files. A new method for initially uploading the file to the server may be implemented in order to allow for larger files to be distributed. Once uploaded to the server there are no known limitations of the system that prevent larger files from being managed.

6.2 Security

Security as a whole was outside of the scope of this thesis. If this system were to be implemented in a real world environment there would need to be an evaluation of the current methods of distributing files. The following are some of the biggest security concerns which absolutely must be addressed in future work.

6.2.1 Client Security

Once a file is distributed to the client there is currently no security on the file. This means that a malicious user may be able to either modify or delete the file without the system knowing it. If this were to happen it may have unforeseen consequences on the system.

Traditional file permissions such as NTFS read/write permissions may be able to be employed in order to secure files so that users do not have access to the files, but the uTorrent.exe process does.

Other methods may be used such as utilizing a separate partition of the hard drive in order to separate the user from the files being stored on the system.

6.2.2 RSS Security

The current distribution method of utilizing RSS feeds in order to distribute files is inherently insecure. There are no security permissions which prevent unauthorized machines from accessing RSS feeds. This means that anyone that can access the RSS feeds via the network would be able to download torrent files.

An alternative means for instructing machines to download files is needed in order to further secure the system. Client side software modifications may be needed in order to add this type of functionality.

6.3 Authentication

A limitation of the current implementation is that only one user has access to upload and download files from the system. Future work may include implementing a new authentication scheme which allows for multiple users to access the system. This multi-authentication scheme would allow users to only have access to files which they have uploaded. They would be unable to view others' files that are stored within the system.

The implementation of this feature would largely be on the server side. The existing implementation would still allow for these files to be distributed regardless of who they belong to.

6.4 Front End

Currently there is much manual configuration that must be done on both the server and client side.

On the server side there could be an installation tool developed which automates much of the manual process that the user must go through. One idea is to use a series of scripts which are accessible through a web GUI. These scripts would install the files, databases, and set permissions for the system.

A method could be developed which would allow for the automation of client configuration and installation. This would reduce the majority of initial setup time and also make it easier for the installation and addition of more clients into the system.

6.5 Scaling

The current implementation has shown that the bottleneck of the system is currently the network. Scaling was outside of the scope for this thesis and therefore was not considered when the proof of concept was designed and implemented. Future work could address this issue by identifying where additional bottlenecks may lie within the system, as well as how to improve the system in order to handle many users and files simultaneously.

One solution may be to add additional trackers in order to further handle the initial uploading and seeding of files to the system. This may allow more users to access the system simultaneously. A method to keep data consistent between these trackers will need to be implemented.

Other bottlenecks such as the non-optimization of database queries and algorithms may need to be addressed as well. Addressing these issues may give a clearer picture of where the hardware bottlenecks lie within the system.

6.6 Notification System

Currently, if a seed or zone were to become unavailable the system has no way of notifying the user that this has happened. A system could be implemented in order to notify a user when a seed or zone becomes unavailable.

The implementation could utilize client-side software which notifies the server that the client is online. If after a set time interval the client fails to respond, the server could notify the user that a seed has become unavailable. From this the server would also be able to determine if an entire set of seeds, or a zone, becomes unavailable. This notification system would be helpful in determining outages, as well as gauging how much available storage is currently online.

A

addfile.php

Interface to add files to system.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Add File</title>
</head>
<?php
	include ("db_connect.php");
	require_once 'Torrent.php'; //helper class for creating torrents
?>
<body>
<form enctype="multipart/form-data" action="addfile.php" method="post">
File: <input name="file" id="file" type="file"  /><br />
Seeds/Zone: <input type="text" name="seeds" /><br />
Zones: <input type="text" name="zones" /><br />
<input type="submit" value="Upload"  />
</form>

<?php
	////////////////////////////////
	/////////////vars/////////////////
	////////////////////////////////
	$announceURL = "http://192.168.0.46/~rivet/announce.php";


	//handle the upload
	if ($_FILES["file"]["error"] > 0){
	  echo "Error: " . $_FILES["file"]["error"] . "<br />";
	}
	else {

		//file info
		echo "<br><br>Upload: " . $_FILES["file"]["name"] . "<br />";
		echo "Type: " . $_FILES["file"]["type"] . "<br />";
		echo "Size: " . ($_FILES["file"]["size"] / 1024) . " Kb<br />";

		//check to see if exists
		if (file_exists("uploads/" . $_FILES["file"]["name"])){
			echo $_FILES["file"]["name"] . " already exists. ";
```

```
                }
                //1. move uploaded file to uploads/
                //2. create torrent
                //3. Call to tracker interface to finish adding torrent
                else{
                        //1. move file
                        $fileName = $_FILES["file"]["name"];
                        move_uploaded_file($_FILES["file"]["tmp_name"], "uploads/" . $fileName);
                        echo "<br>Stored in: " . "uploads/" . $fileName ."<br>";

                        //2. create torrent file
                        $torrent = new Torrent( './uploads/'.$fileName, $announceURL );
                        if ( ! $error = $torrent->error() ) // error method return the last error message
                                $torrent->save('torrents/'.$fileName.'.torrent'); // save to disk
                        else
                                echo '<br>DEBUG: ',$error;


                        echo "<br><br>Torrent file created: torrents/".$fileName.".torrent<br>";
                        echo "Hash: " . $torrent->hash_info();

                        //3. call tracker interface
                        exec('php tracker_interface.php "torrents/'.$fileName.'.torrent"
"http://192.168.0.46/~joey/uploads/'.$fileName.'"');

                        //4. add to master_db
                        $hash = $torrent->hash_info();
                        $seeds = $_POST['seeds'];
                        $zones = $_POST['zones'];
                        $query = "INSERT into files (filename, file_id, seeds, zones) VALUES ('$fileName',
'$hash', '$seeds', '$zones')";
                        mysql_query($query);

                        echo '<br>Great Success?!';
                }
        }
?>
</body>
</html>
```

B

addpeer.php

Interface to add peers to system.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Add Peers</title>
</head>
<?php
 include("db_connect.php");
?>

<body>

<?php
//If Blank Echo
if ($_GET['ip'] == ''){
	echo 'Add Peer:<br>';
}
//Add Peers
else {
	//vars
	$ip = $_GET['ip'];
	$zone = $_GET['zone'];
	$space = $_GET['space'];

	// add peer to peer table
	$sql = "INSERT into peers (ip,zone,space) VALUES ('$ip','$zone','$space')";
	mysql_query($sql) or die(mysql_error());

	//create table for IP to store files peer should download
	$hyphenIP = str_replace( ".", "-", "$ip"); //mysql doesn't allow . in table names :(
	$sql = "CREATE TABLE `$hyphenIP` (
					filename VARCHAR(255) NOT NULL,
					file_id VARCHAR(255) NOT NULL PRIMARY KEY
			)";
	mysql_query($sql) or die(mysql_error());

	//create ./rss/<ip>/ folder for rss feed
	exec("mkdir ./rss/$ip && chmod 777 ./rss/$ip");
```

```
}
?>

<form action="addpeer.php" method="get">
IP: <input type="text" name="ip"  /><br />
ZONE: <input type="text" name="zone"  /><br />
SPACE(MB): <input type="text" name="space"  /><br />
<input type="submit" />
</form>

<br /><br />
Peers:<br />
<?php
	//LIST PEERS
	$sql = "SELECT ip,zone,space FROM peers WHERE 1=1";
	$result = mysql_query ($sql) or die(mysql_error());
	while ($row = mysql_fetch_assoc($result)){
		echo "IP: ".$row['ip']." ZONE: ".$row['zone']." SPACE: ".$row['space']."<br>";
	}
?>

</body>
</html>
```

C

db_connect.php

Included in many other scripts to connect to database.

```
<?php
	//some vars to make things easier?
	$db_user = 'root';
	$db_password = 'netsys';
	$db_address = 'localhost';

	//connect to the server
	if ( !$link = mysql_connect ($db_address, $db_user, $db_password) ){
		die('Could not connect to SERVER' . $db_address);
	}

	//select db
	if ( ! mysql_select_db ('master_db') ){
		die("Could not open database");
	}
?>
```

D

refresh_all.php

Script that calls update_file.php for all files in database.

```
<?php
	/*
		Nifty script that calls update_file.php for all files in master_db.
		Finally. Done?
	*/

	include("db_connect.php");
	$sql = "SELECT file_id FROM files"; // select all file_ids
	$query = mysql_query($sql) or die(mysql_error());

	//For every file_id run update_file.php <file_id>
	while ( $row = mysql_fetch_row($query) ){
		exec("php update_file.php $row[0]");
	}
?>
```

E

rss_create.php

Script used to create an RSS feed for a given IP.

```
<?php
	/*
		Create RSS Feed given IP
		Usage: php rss_create.php <ip>
		Output: ./rss/<ip>/rss.xml
	*/

	//some vars to make things easier?
	$db_user = 'root';
	$db_password = 'netsys';
	$db_address = 'localhost';

	//connect to the server twice - once for each db
	$linkMaster = mysql_connect ($db_address, $db_user, $db_password);
	$linkTracker = mysql_connect ($db_address, $db_user, $db_password, true);

	//select dbs
	mysql_select_db ('master_db',$linkMaster); //master db
	mysql_select_db ('rivet', $linkTracker); //torrent db

	//replace . with - in IP
	$hyphenIP = str_replace(".","-",$argv[1]);

	//select all file_ids IP should have
	$sql = "SELECT file_id,filename FROM `$hyphenIP`";
	$query = mysql_query($sql, $linkMaster) or die(mysql_error());

	//open ./rss/<ip>/rss.xml for writing
	$fd = fopen("./rss/$argv[1]/rss.xml", "w") or die("Error: Unable to write to ./rss/$argv[1]/rss.xml
file!");

	//starting RSS info
	$start_text =
	"<?xml version=\"1.0\" encoding=\"utf-8\"?>\n" .
	"<rss version=\"2.0\">\n" .
	"<channel>\n" .
	"<title></title>\n" .
	"<link></link>\n" .
	"<description></description>\n" .
	"<lastBuildDate>" . date('D, j M Y h:i:s') . " -0500</lastBuildDate>\n";
```

```
	$middle_text=""; //for rss items

	//for ever file_id generate rss feed info
	while ($therow = mysql_fetch_row($query)){
		//url
		$url = "http://192.168.0.46/~rivet/torrents/$therow[1].torrent";

		//info for rss
		$sql = "SELECT filename,url,size,pubDate FROM namemap WHERE info_hash =
'$therow[0]'";
		$fileInfo = mysql_query($sql, $linkTracker);

		//create rss item
		while ($row = mysql_fetch_row($fileInfo) ){
			$middle_text = $middle_text . "<item>\n" .
			"<title>" . $row[0] . "</title>\n" .
			"<description>" . $row[0] . "</description>\n" .
			"<pubDate>" . $row[3] . " -0500</pubDate>\n" .
			"<guid>" . $url . "</guid>\n" .
			"<link>" . $url . "</link>\n" .
			"<enclosure url=\"" . $url . "\" type=\"application/x-bittorrent\" />\n" .
			"</item>\n";
		}
	}

	//close rss and write
	$end_text = "</channel>\n</rss>";
	fwrite($fd, $start_text . $middle_text . $end_text);
	fclose($fd);
?>
```

F

Torrent.php

Class used to create .torrent files.

Author: Adrien Gibrat

```
<?php
/**
 * Torrent
 *
 * PHP version 5.2+ (with cURL extention enabled)
 *
 * LICENSE: This source file is subject to version 3 of the GNU GPL
 * that is available through the world-wide-web at the following URI:
 * http://www.gnu.org/licenses/gpl.html. If you did not receive a copy of
 * the GNU GPL License and are unable to obtain it through the web, please
 * send a note to adrien.gibrat@gmail.com so I can mail you a copy.
 *
 * 1) Features:
 * - Decode torrent file or data from local file and distant url
 * - Build torrent from source folder/file(s) or distant url
 * - Super easy usage & syntax
 * - Silent Exception error system
 *
 * 2) Usage example
 * <code>
require_once 'Torrent.php';

// create torrent
$torrent = new Torrent( './path-to-file-or-folder', 'http://torrent.tracker/annonce' );
if ( ! $error = $torrent->error() ) // error method return the last error message
        $torrent->save('test.torrent'); // save to disk
else
        echo '<br>DEBUG: ',$error;

// print torrent infos
$torrent = new Torrent( './test.torrent' );
echo '<pre>private: ', $torrent->is_private() ? 'yes' : 'no',
         '<br>annonce: ';
var_dump( $torrent->announce() );
echo '<br>name: ', $torrent->name(),
         '<br>comment: ', $torrent->comment(),
         '<br>piece_length: ', $torrent->piece_length(),
         '<br>size: ', $torrent->size( 2 ),
         '<br>hash info: ', $torrent->hash_info(),
         '<br>stats: ';
```

```
var_dump( $torrent->scrape() );
echo '<br>content: ';
var_dump( $torrent->content() );
echo '<br>source: ',
        $torrent;

// modify torrent
$torrent->announce('http://alternate-torrent.tracker/annonce'); // add a tracker
$torrent->announce(false); // reset announce trackers
$torrent->announce(array('http://torrent.tracker/annonce', 'http://alternate-
torrent.tracker/annonce')); // set tracker(s), it also works with a 'one tracker' array...
$torrent->announce(array(array('http://torrent.tracker/annonce', 'http://alternate-
torrent.tracker/annonce'), 'http://another-torrent.tracker/annonce')); // set tiered trackers
$torrent->comment('hello world');
$torrent->name('test torrent');
$torrent->is_private(true);
$torrent->httpseeds('http://file-hosting.domain/path/'); // Bittornado implementation
$torrent->url_list(array('http://file-hosting.domain/path/','http://another-file-hosting.domain/path/'));
// GetRight implementation

// print errors
if ( $errors = $torrent->errors() ) // errors method return the error stack
        var_dump( '<br>DEBUG: ', $errors );

// send to user
//$torrent->send();
 * </code>
 *
 * @author   Adrien Gibrat <adrien.gibrat@gmail.com>
 * @tester   Jeong, official tester ;) Thanks for your precious feedback
 * @copyleft 2010 - Just use it!
 * @license  http://www.gnu.org/licenses/gpl.html GNU General Public License version 3
 * @version  Release: 1.2 (6 july 2010)
 */
class Torrent {

        /**
        * @const float Default http timeout
        */
        const timeout = 30;

        /**
        * @var array List of error occured
        */
        static protected $errors = array();

        /** Read and decode torrent file/data OR build a torrent from source folder/file(s)
         * Supported signatures:
```

```
	 * - Torrent(); // get an instance (usefull to scrape and check errors)
	 * - Torrent( string $torrent ); // analyse a torrent file
	 * - Torrent( string $torrent, string $announce );
	 * - Torrent( string $torrent, array $meta );
	 * - Torrent( string $file_or_folder ); // create a torrent file
	 * - Torrent( string $file_or_folder, string $announce_url, [int $piece_length] );
	 * - Torrent( string $file_or_folder, array $meta, [int $piece_length] );
	 * - Torrent( array $files_list );
	 * - Torrent( array $files_list, string $announce_url, [int $piece_length] );
	 * - Torrent( array $files_list, array $meta, [int $piece_length] );
	 * @param string|array torrent to read or source folder/file(s) (optional, to get an instance)
	 * @param string|array announce url or meta informations (optional)
	 * @param int piece length (optional)
	 */
	public function __construct ( $data = null, $meta = array(), $piece_length = 256 ) {
		if ( is_null( $data ) )
			return false;
		if ( $piece_length < 32 || $piece_length > 4096 )
			return self::set_error( new Exception( 'Invalid piece lenth, must be between 32
and 4096' ) );
		if ( is_string( $meta ) )
			$meta = array( 'announce' => $meta );
		if ( $this->build( $data, $piece_length * 1024 ) )
			$this->touch();
		else
			$meta = array_merge( $meta, $this->decode( $data ) );
		foreach( $meta as $key => $value )
			$this->{$key} = $value;
	}

	/** Convert the current Torrent instance in torrent format
	 * @return string encoded torrent data
	 */
	public function __toString() {
		return $this->encode( $this );
	}

	/** Return last error message
	 * @return string|boolean last error message or false if none
	 */
	public function error() {
		return empty( self::$errors ) ?
			false :
			self::$errors[0]->getMessage();
	}

	/** Return Errors
	 * @return array|boolean error list or false if none
```

```
	 */
	public function errors() {
		return empty( self::$errors ) ?
			false :
			self::$errors;
	}

	/**** Getters and setters ****/

	/** Getter and setter of torrent announce url / list
	 * If the argument is a string, announce url is added to announce list (or set as announce if announce is not set)
	 * If the argument is an array/object, set announce url (with first url) and list (if array has more than one url), tiered list supported
	 * If the argument is false announce url & list are unset
	 * @param null|false|string|array announce url / list, reset all if false (optional, if omitted it's a getter)
	 * @return string|array|null announce url / list or null if not set
	 */
	public function announce ( $announce = null ) {
		if ( is_null( $announce ) )
			return ! isset( $this->{'announce-list'} ) ?
				isset( $this->announce ) ? $this->announce : null :
				$this->{'announce-list'};
		$this->touch();
		if ( is_string( $announce ) && isset( $this->announce ) )
			return $this->{'announce-list'} = self::announce_list( isset( $this->{'announce-list'} ) ? $this->{'announce-list'} : $this->announce, $announce );
		unset( $this->{'announce-list'} );
		if ( is_array( $announce ) || is_object( $announce ) )
			if ( ( $this->announce = self::first_announce( $announce ) ) && count( $announce ) > 1 )
				return $this->{'announce-list'} = self::announce_list( $announce );
			else
				return $this->announce;
		if ( ! isset( $this->announce ) && $announce )
			return $this->announce = (string) $announce;
		unset( $this->announce );
	}

	/** Getter and setter of torrent comment
	 * @param null|string comment (optional, if omitted it's a getter)
	 * @return string|null comment or null if not set
	 */
	public function comment ( $comment = null ) {
		return is_null( $comment ) ?
			isset( $this->comment ) ? $this->comment : null :
			$this->touch( $this->comment = (string) $comment );
```

```
}

/** Getter and setter of torrent name
 * @param null|string name (optional, if omitted it's a getter)
 * @return string|null name or null if not set
 */
public function name ( $name = null ) {
	return is_null( $name ) ?
		isset( $this->info['name'] ) ? $this->info['name'] : null :
		$this->touch( $this->info['name'] = (string) $name );
}

/** Getter and setter of private flag
 * @param null|boolean is private or not (optional, if omitted it's a getter)
 * @return boolean private flag
 */
public function is_private ( $private = null ) {
	return is_null( $private ) ?
		! empty( $this->info['private'] ) :
		$this->touch( $this->info['private'] = $private ? 1 : 0 );
}

/** Getter and setter of webseed(s) url list ( GetRight implementation )
 * @param null|string|array webseed or webseeds mirror list (optional, if omitted it's a getter)
 * @return string|array|null webseed(s) or null if not set
 */
public function url_list ( $urls = null ) {
	return is_null( $urls ) ?
		isset( $this->{'url-list'} ) ? $this->{'url-list'} : null :
		$this->touch( $this->{'url-list'} = is_string( $urls) ? $urls : (array) $urls );
}

/** Getter and setter of httpseed(s) url list ( Bittornado implementation )
 * @param null|string|array httpseed or httpseeds mirror list (optional, if omitted it's a getter)
 * @return array|null httpseed(s) or null if not set
 */
public function httpseeds ( $urls = null ) {
	return is_null( $urls ) ?
		isset( $this->httpseeds ) ? $this->httpseeds : null :
		$this->touch( $this->httpseeds = (array) $urls );
}

/**** Analyze BitTorrent ****/

/** Get piece length
 * @return integer piece length or null if not set
 */
public function piece_length () {
```

```
		return isset( $this->info['piece length'] ) ?
			$this->info['piece length'] :
			null;
	}

	/** Compute hash info
	 * @return string hash info or null if info not set
	 */
	public function hash_info () {
		return isset( $this->info ) ?
			sha1( self::encode( $this->info ) ) :
			null;
	}

	/** List torrent content
	 * @param integer|null size precision (optional, if omitted returns sizes in bytes)
	 * @return array file(s) and size(s) list, files as keys and sizes as values
	 */
	public function content ( $precision = null ) {
		$files = array();
		if ( isset( $this->info['files'] ) && is_array( $this->info['files'] ) )
			foreach ( $this->info['files'] as $file )
				$files[self::path( $file['path'], $this->info['name'] )] = $precision ?
					self::format( $file['length'], $precision ) :
					$file['length'];
		elseif ( isset( $this->info['name'] ) )
				$files[$this->info['name']] = $precision ?
					self::format( $this->info['length'], $precision ) :
					$this->info['length'];
		return $files;
	}

	/** List torrent content pieces and offset(s)
	 * @return array file(s) and pieces/offset(s) list, file(s) as keys and pieces/offset(s) as values
	 */
	public function offset () {
		$files = array();
		$size = 0;
		if ( isset( $this->info['files'] ) && is_array( $this->info['files'] ) )
			foreach ( $this->info['files'] as $file )
				$files[self::path( $file['path'], $this->info['name'] )] = array(
					'startpiece'	=> floor( $size / $this->info['piece length'] ),
					'offset'  => fmod( $size, $this->info['piece length'] ),
					'size'		=> $size += $file['length'],
					'endpiece'	=> floor( $size / $this->info['piece length'] )
				);
		elseif ( isset( $this->info['name'] ) )
				$files[$this->info['name']] = array(
```

```
                                        'startpiece'        => 0,
                                        'offset'  => 0,
                                        'size'              => $this->info['length'],
                                        'endpiece'          => floor( $this->info['length'] / $this->info['piece
length'] )
                                );
                return $files;
        }

        /** Sum torrent content size
         * @param integer|null size precision (optional, if omitted returns size in bytes)
         * @return integer|string file(s) size
         */
        public function size ( $precision = null ) {
                $size = 0;
                if ( isset( $this->info['files'] ) && is_array( $this->info['files'] ) )
                        foreach ( $this->info['files'] as $file )
                                $size += $file['length'];
                elseif ( isset( $this->info['name'] ) )
                                $size = $this->info['length'];
                return is_null( $precision ) ?
                        $size :
                        self::format( $size, $precision );
        }

        /** Request torrent statistics from scrape page USING CURL!!
         * @param string|array announce or scrape page url (optional, to request an alternative tracker
BUT requirered for static call)
         * @param string torrent hash info (optional, requirered ONLY for static call)
         * @param float read timeout in seconds (optional, default to self::timeout 30s)
         * @return array tracker torrent statistics
         */
        /* static */ public function scrape ( $announce = null, $hash_info = null, $timeout = self::timeout
) {
                $packed_hash = urlencode( pack('H*', $hash_info ? $hash_info : $this->hash_info() ) );
                $handles = $scrape = array();
                if ( ! function_exists( 'curl_multi_init' ) )
                        return self::set_error( new Exception( 'Install CURL with "curl_multi_init"
enabled' ) );
                $curl = curl_multi_init();
                foreach ( (array) ($announce ? $announce : $this->announce()) as $tier )
                        foreach ( (array) $tier as $tracker ) {
                                $tracker = str_ireplace( array( 'udp://', '/announce', ':80/' ), array(
'http://', '/scrape', '/' ), $tracker );
                                if ( isset( $handles[$tracker] ) )
                                        continue;
                                $handles[$tracker] = curl_init( $tracker . '?info_hash=' . $packed_hash );
                                curl_setopt( $handles[$tracker], CURLOPT_RETURNTRANSFER, true );
```

```
                                curl_setopt( $handles[$tracker], CURLOPT_TIMEOUT, $timeout );
                                curl_multi_add_handle( $curl, $handles[$tracker] );
                        }
                do {
        while ( ( $state = curl_multi_exec( $curl, $running ) ) == CURLM_CALL_MULTI_PERFORM );
        if( $state != CURLM_OK )
           continue;
                        while ( $done = curl_multi_info_read( $curl ) ) {
                                $info = curl_getinfo( $done['handle'] );
          $tracker = array_shift( explode( '?', $info['url'], 2 ) );
          if ( empty( $info['http_code'] ) )
                continue $scrape[$tracker] = self::set_error( new Exception( 'Tracker request timeout ('
. $timeout . 's)' ), true );
          elseif ( $info['http_code'] != 200 )
                continue $scrape[$tracker] = self::set_error( new Exception( 'Tracker request failed (' .
$info['http_code'] . ' code)' ), true );
                                $stats = self::decode_data( curl_multi_getcontent( $done['handle']  ) );
                                curl_multi_remove_handle( $curl, $done['handle'] );
                                $scrape[$tracker] = empty( $stats['files'] ) ?
                                        self::set_error( new Exception( 'Empty scrape data' ), true ) :
                                        array_shift( $stats['files'] ) + ( empty( $stats['flags'] ) ? array() :
$stats['flags'] );
                        }
                } while ( $running );
                curl_multi_close( $curl );
                return $scrape;
          }

          /**** Save and Send ****/

          /** Save torrent file to disk
           * @param null|string name of the file (optional)
           * @return boolean file has been saved or not
           */
          public function save ( $filename = null ) {
                return file_put_contents( is_null( $filename ) ? $this->info['name'] . '.torrent' :
$filename, $this->encode( $this ) );
          }

          /** Send torrent file to client
           * @param null|string name of the file (optional)
           * @return void script exit
           */
          public function send ( $filename = null ) {
                $data = $this->encode( $this );
                header( 'Content-type: application/x-bittorrent' );
                header( 'Content-Length: ' . strlen( $data ) );
```

```
                header( 'Content-Disposition: attachment; filename="' . ( is_null( $filename ) ? $this->info['name'] . '.torrent' : $filename ) . '"' );
                exit( $data );
        }

        /**** Encode BitTorrent ****/

        /** Encode torrent data
         * @param mixed data to encode
         * @return string torrent encoded data
         */
        static public function encode ( $mixed ) {
                switch ( gettype( $mixed ) ) {
                        case 'integer':
                        case 'double':
                                return self::encode_integer( $mixed );
        case 'object':
         $mixed = (array) $mixed; //Thanks to W-Shadow: http://w-shadow.com/blog/2008/11/11/parse-edit-and-create-torrent-files-with-php/
        case 'array':
                                return self::encode_array( $mixed );
                        default:
                                return self::encode_string( (string) $mixed );
                }
        }

        /** Encode torrent string
         * @param string string to encode
         * @return string encoded string
         */
        static private function encode_string ( $string ) {
                return strlen( $string ) . ':' . $string;
        }

        /** Encode torrent integer
         * @param integer integer to encode
         * @return string encoded integer
         */
        static private function encode_integer ( $integer ) {
                return 'i' . $integer . 'e';
        }

        /** Encode torrent dictionary or list
         * @param array array to encode
         * @return string encoded dictionary or list
         */
        static private function encode_array ( $array ) {
                if ( self::is_list( $array ) ) {
```

```
                $return = 'l';
                foreach ( $array as $value )
                        $return .= self::encode( $value );
        } else {
                ksort( $array, SORT_STRING );
                $return = 'd';
                foreach ( $array as $key => $value )
                        $return .= self::encode( strval( $key ) ) . self::encode( $value );
        }
        return $return . 'e';
}

/**** Decode BitTorrent ****/

/** Decode torrent data or file
 * @param string data or file path to decode
 * @return array decoded torrent data
 */
static protected function decode ( $string ) {
        $data = is_file( $string ) || self::url_exists( $string ) ?
                self::file_get_contents( $string ) :
                $string;
        return (array) self::decode_data( $data );
}

/** Decode torrent data
 * @param string data to decode
 * @return array decoded torrent data
 */
static private function decode_data ( & $data ) {
        switch( self::char( $data ) ) {
        case 'i':
                $data = substr( $data, 1 );
                return self::decode_integer( $data );
        case 'l':
                $data = substr( $data, 1 );
                return self::decode_list( $data );
        case 'd':
                $data = substr( $data, 1 );
                return self::decode_dictionary( $data );
        default:
                return self::decode_string( $data );
        }
}

/** Decode torrent dictionary
 * @param string data to decode
 * @return array decoded dictionary
```

```
 */
static private function decode_dictionary ( & $data ) {
	$dictionary = array();
	$previous = null;
	while ( ( $char = self::char( $data ) ) != 'e' ) {
		if ( $char === false )
			return self::set_error( new Exception( 'Unterminated dictionary' ) );
		if ( ! ctype_digit( $char ) )
			return self::set_error( new Exception( 'Invalid dictionary key' ) );
		$key = self::decode_string( $data );
		if ( isset( $dictionary[$key] ) )
			return self::set_error( new Exception( 'Duplicate dictionary key' ) );
		if ( $key < $previous )
			return self::set_error( new Exception( 'Missorted dictionary key' ) );
		$dictionary[$key] = self::decode_data( $data );
		$previous = $key;
	}
	$data = substr( $data, 1 );
	return $dictionary;
}

/** Decode torrent list
 * @param string data to decode
 * @return array decoded list
 */
static private function decode_list ( & $data ) {
	$list = array();
	while ( ( $char = self::char( $data ) ) != 'e' ) {
		if ( $char === false )
			return self::set_error( new Exception( 'Unterminated list' ) );
		$list[] = self::decode_data( $data );
	}
	$data = substr( $data, 1 );
	return $list;
}

/** Decode torrent string
 * @param string data to decode
 * @return string decoded string
 */
static private function decode_string ( & $data ) {
	if ( self::char( $data ) === '0' && substr( $data, 1, 1 ) != ':' )
		self::set_error( new Exception( 'Invalid string length, leading zero' ) );
	if ( ! $colon = @strpos( $data, ':' ) )
		return self::set_error( new Exception( 'Invalid string length, colon not found' ) );
	$length = intval( substr( $data, 0, $colon ) );
	if ( $length + $colon + 1 > strlen( $data ) )
```

```
                        return self::set_error( new Exception( 'Invalid string, input too short for string
length' ) );
                $string = substr( $data, $colon + 1, $length );
                $data = substr( $data, $colon + $length + 1 );
                return $string;
        }

        /** Decode torrent integer
         * @param string data to decode
         * @return integer decoded integer
         */
        static private function decode_integer ( & $data ) {
                $start = 0;
                $end     = strpos( $data, 'e');
                if ( $end === 0 )
                        self::set_error( new Exception( 'Empty integer' ) );
                if ( self::char( $data ) == '-' )
                        $start++;
                if ( substr( $data, $start, 1 ) == '0' && ( $start != 0 || $end > $start + 1 ) )
                        self::set_error( new Exception( 'Leading zero in integer' ) );
                if ( ! ctype_digit( substr( $data, $start, $end ) ) )
                        self::set_error( new Exception( 'Non-digit characters in integer' ) );
                $integer = substr( $data, 0, $end );
                $data = substr( $data, $end + 1 );
                return $integer + 0;
        }

        /**** Internal Helpers ****/

        /** Build torrent info
         * @param string|array source folder/file(s) path
         * @param integer piece length
         * @return array|boolean torrent info or false if data isn't folder/file(s)
         */
        protected function build ( $data, $piece_length ) {
                if ( is_null( $data ) )
                        return false;
                elseif ( is_array( $data ) && self::is_list( $data ) )
                        return $this->info = $this->files( $data, $piece_length );
                elseif ( is_dir( $data ) )
                        return $this->info = $this->folder( $data, $piece_length );
                elseif ( ( is_file( $data ) || self::url_exists( $data ) ) && ! self::is_torrent( $data ) )
                        return $this->info = $this->file( $data, $piece_length );
                else
                        return false;
        }

        /** Set torrent creator and creation date
```

```
		 * @param any param
		 * @return any param
		 */
		protected function touch ( $void = null ) {
			$this->{'created by'}		= 'Torrent PHP Class - Adrien Gibrat';
			$this->{'creation date'}	= time();
			return $void;
		}

		/** Add an error to errors stack
		 * @param Exception error to add
		 * @param boolean return error message or not (optional, default to false)
		 * @return boolean|string return false or error message if requested
		 */
		static protected function set_error ( $exception, $message = false ) {
			return ( array_unshift( self::$errors,  $exception ) && $message ) ? $exception->getMessage() : false;
		}

		/** Build announce list
		 * @param string|array announce url / list
		 * @param string|array announce url / list to add (optionnal)
		 * @return array announce list (array of arrays)
		 */
		static protected function announce_list( $announce, $merge = array() ) {
			return array_map( create_function( '$a', 'return (array) $a;' ), array_merge( (array) $announce, (array) $merge ) );
		}

		/** Get the first announce url in a list
		 * @param array announce list (array of arrays if tiered trackers)
		 * @return string first announce url
		 */
		static protected function first_announce( $announce ) {
			while ( is_array( $announce ) )
				$announce = reset( $announce );
			return $announce;
		}

		/** Helper to pack data hash
		 * @param string data
		 * @return string packed data hash
		 */
		static protected function pack ( & $data ) {
			return pack('H*', sha1( $data ) ) . ( $data = null );
		}

		/** Helper to build file path
```

```
 * @param array file path
 * @param string base folder
 * @return string real file path
 */
static protected function path ( $path, $folder ) {
	array_unshift( $path, $folder );
	return join( DIRECTORY_SEPARATOR, $path );
}

/** Helper to test if an array is a list
 * @param array array to test
 * @return boolean is the array a list or not
 */
static protected function is_list ( $array ) {
	foreach ( array_keys( $array ) as $key )
		if ( ! is_int( $key ) )
			return false;
	return true;
}

/** Build pieces depending on piece length from a file handler
 * @param ressource file handle
 * @param integer piece length
 * @param boolean is last piece
 * @return string pieces
 */
private function pieces ( $handle, $piece_length, $last = true ) {
	static $piece, $length;
	if ( empty( $length ) )
		$length = $piece_length;
	$pieces = null;
	while ( ! feof( $handle ) ) {
		if ( ( $length = strlen( $piece .= fread( $handle, $length ) ) ) == $piece_length )
			$pieces .= self::pack( $piece );
		elseif ( ( $length = $piece_length - $length ) < 0 )
			return self::set_error( new Exception( 'Invalid piece length!' ) );
	}
	fclose( $handle );
	return $pieces . ( $last && $piece ? self::pack( $piece ) : null);
}

/** Build torrent info from single file
 * @param string file path
 * @param integer piece length
 * @return array torrent info
 */
private function file ( $file, $piece_length ) {
	if ( ! $handle = self::fopen( $file, $size = self::filesize( $file ) ) )
```

```
                        return self::set_error( new Exception( 'Failed to open file: "' . $file . '"' ) );
                return array(
                        'length' => $size,
                        'name'          => end( explode( DIRECTORY_SEPARATOR, $file ) ),
                        'piece length'  => $piece_length,
                        'pieces' => $this->pieces( $handle, $piece_length )
                );
        }

        /** Build torrent info from files
         * @param array file list
         * @param integer piece length
         * @return array torrent info
         */
        private function files ( $files, $piece_length ) {
                $files = array_map( 'realpath', $files );
                sort( $files );
                usort( $files, create_function( '$a,$b', 'return strrpos($a,DIRECTORY_SEPARATOR)-
strrpos($b,DIRECTORY_SEPARATOR);' ) );
                $path   = explode( DIRECTORY_SEPARATOR, dirname( realpath( current( $files ) ) ) );
                $pieces = null; $info_files = array(); $count = count($files) - 1;
                foreach ( $files as $i => $file ) {
                        if ( $path != array_intersect_assoc( $file_path = explode(
DIRECTORY_SEPARATOR, $file ), $path ) )
                                continue self::set_error( new Exception( 'Files must be in the same
folder: "' . $file . '" discarded' ) );
                        if ( ! $handle = self::fopen( $file, $filesize = self::filesize( $file ) ) )
                                continue self::set_error( new Exception( 'Failed to open file: "' . $file . '"
discarded' ) );
                        $pieces .= $this->pieces( $handle, $piece_length, $count == $i );
                        $info_files[] = array(
                                'length' => $filesize,
                                'path'          => array_diff( $file_path, $path )
                        );
                }
                return array(
                        'files'         => $info_files,
                        'name'          => end( $path ),
                        'piece length'  => $piece_length,
                        'pieces' => $pieces
                );

        }

        /** Build torrent info from folder content
         * @param string folder path
         * @param integer piece length
         * @return array torrent info
```

```
	 */
	private function folder ( $dir, $piece_length ) {
		return $this->files( self::scandir( $dir ), $piece_length );
	}

	/** Helper to return the first char of encoded data
	 * @param string encoded data
	 * @return string|boolean first char of encoded data or false if empty data
	 */
	static private function char ( $data ) {
		return empty( $data ) ?
			false :
			substr( $data, 0, 1 );
	}

	/**** Public Helpers ****/

	/** Helper to format size in bytes to human readable
	 * @param integer size in bytes
	 * @param integer precision after coma
	 * @return string formated size in appropriate unit
	 */
	static public function format ( $size, $precision = 2 ) {
		$units = array ('octets', 'Ko', 'Mo', 'Go', 'To');
		while( ( $next = next( $units ) ) && $size > 1024 )
			$size /= 1024;
		return round( $size, $precision ) . ' ' . ( $next ? prev( $units ) : end( $units ) );
	}

	/** Helper to return filesize (even bigger than 2Gb -linux only- and distant files size)
	 * @param string file path
	 * @return double|boolean filesize or false if error
	 */
	static public function filesize ( $file ) {
		if ( is_file( $file ) )
			return (double) sprintf( '%u', @filesize( $file ) );
		else if ( $content_length = preg_grep( $pattern = '#^Content-Length:\s+(\d+)$#i', (array)
@get_headers( $file ) ) )
			return (int) preg_replace( $pattern, '$1', reset( $content_length ) );
	}

	/** Helper to open file to read (even bigger than 2Gb, linux only)
	 * @param string file path
	 * @param integer|double file size (optional)
	 * @return ressource|boolean file handle or false if error
	 */
	static public function fopen ( $file, $size = null ) {
		if ( ( is_null( $size ) ? self::filesize( $file ) : $size ) <= 2 * pow( 1024, 3 ) )
```

```
                return fopen( $file, 'r' );
        elseif ( PHP_OS != 'Linux' )
                return self::set_error( new Exception( 'File size is greater than 2GB. This is only
supported under Linux' ) );
        elseif ( ! is_readable( $file ) )
                return false;
        else
                return popen( 'cat ' . escapeshellarg( realpath( $file ) ), 'r' );
    }

    /** Helper to scan directories files and sub directories recursivly
     * @param string directory path
     * @return array directory content list
     */
    static public function scandir ( $dir ) {
        $paths = array();
        foreach ( scandir( $dir ) as $item )
                if ( $item != '.' && $item != '..' )
                        if ( is_dir( $path = realpath( $dir . DIRECTORY_SEPARATOR .
$item ) ) )
                                $paths = array_merge( self::scandir( $path ), $paths );
                        else
                                $paths[] = $path;
        return $paths;
    }

    /** Helper to check if url exists
     * @param string url to check
     * @return boolean does the url exist or not
     */
    static public function url_exists ( $url ) {
        return preg_match( '#^http(s)?://[a-z0-9-]+(.[a-z0-9-]+)*(:[0-9]+)?(/.*)?$#i', $url ) ?
                (bool) preg_grep( '#^HTTP/.*\s(200|304)\s#', (array) @get_headers( $url ) ) :
                false;
    }

    /** Helper to check if a file is a torrent
     * @param string file location
     * @param float http timeout (optional, default to self::timeout 30s)
     * @return boolean is the file a torrent or not
     */
    static public function is_torrent ( $file, $timeout = self::timeout ) {
        return self::file_get_contents( $file, $timeout, 0, 11 ) === 'd8:announce';
    }

    /** Helper to get (distant) file content
     * @param string file location
     * @param float http timeout (optional, default to self::timeout 30s)
```

```
		 * @param integer starting offset (optional, default to null)
		 * @param integer content length (optional, default to null)
		 * @return string|boolean file content or false if error
		 */
		static public function file_get_contents ( $file, $timeout = self::timeout, $offset = null, $length =
null ) {
			if ( is_file( $file ) || ini_get( 'allow_url_fopen' ) ) {
				$context = ! is_file( $file ) && $timeout ?
					stream_context_create( array( 'http' => array( 'timeout' => $timeout ) ) )
:
					null;
				return ! is_null( $offset ) ? $length ?
					@file_get_contents( $file, false, $context, $offset, $length ) :
					@file_get_contents( $file, false, $context, $offset ) :
					@file_get_contents( $file, false, $context );
			} elseif ( ! function_exists( 'curl_init' ) )
				return self::set_error( new Exception( 'Install CURL or enable "allow_url_fopen"'
) );
			$handle = curl_init( $file );
			if ( $timeout )
				curl_setopt( $handle, CURLOPT_TIMEOUT, $timeout );
			if ( $offset || $length )
				curl_setopt( $handle, CURLOPT_RANGE, $offset . '-' . ( $length ? $offset +
$length -1 : null ) );
			curl_setopt( $handle, CURLOPT_RETURNTRANSFER, 1 );
			$content = curl_exec( $handle );
			$size = curl_getinfo( $handle, CURLINFO_CONTENT_LENGTH_DOWNLOAD );
			curl_close( $handle );
			return ( $offset && $size == -1 ) || ( $length && $length != $size ) ? $length ?
				substr( $content, $offset, $length) :
				substr( $content, $offset) :
				$content;
		}

}

?>
```

G

tracker_interface.php

Script used to authenticate and add files to BitTorrent tracker.

```
<?php
/*
	Add files to tracker.
	Usage: php tracker_interface.php <torrentLocation> <httpSeedLocation>
*/
				//authenticate w tracker
				$ch = curl_init();
				curl_setopt($ch, CURLOPT_URL, "http://192.168.0.46/~rivet/login.php"); //login
url
				curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1); //return as a string
				curl_setopt($ch, CURLOPT_POST, true); //POST
				curl_setopt($ch, CURLOPT_COOKIEJAR, './cookies/cookies.txt'); //save cookies
				curl_setopt($ch, CURLOPT_COOKIEFILE, './cookies/cookies.txt'); //use cookies
				curl_setopt($ch, CURLOPT_FOLLOWLOCATION, 1); //follow redirects

				//form login data
				$data = array(
					'f_user' => 'joey',
					'f_pass' => 'netsys',
					'LogIn' => 'Log+In',
					'legalterms' => 'on'
				);

				curl_setopt($ch, CURLOPT_POSTFIELDS, $data); // form data

			curl_exec($ch); //execute login

				//add torrent
				curl_setopt($ch, CURLOPT_URL, "http://192.168.0.46/~rivet/newtorrents.php");
//add url

				//form torrent data
				$torrentLoc = $argv[1];
				$httpseedLoc = $argv[2];

				//torrent data
				$data2 = array (
					'torrent' => "@$torrentLoc",
					'getrightseed' => 'enabled',
					'httpftplocation' => "$httpseedLoc",
					'autoset' => 'enabled'
```

```
                    );
                    curl_setopt($ch, CURLOPT_POSTFIELDS, $data2);//torrent data
                    curl_exec($ch); //submit torrent
                    curl_close($ch); //close to save lulsources
?>
```

H

update_file.php

Script used to ensure a file is present on enough seeds/zone and in enough zones.

```
<?php
	/*
		1. Use file_id to ensure enough seeds / zones seeding.
		2. Add additional seeds or zones if needed.
		2b. Determine which seeds to be added.

		Usage: php update_file <file_id>

		1048576 bytes in 1 megabyte
	*/

	// LIMITATION, ONLY WORKS IF IP EXISTS IN EVERY ZONE STARING WITH 1 - $numZones
	$numZones = 4;

	/*
		Adds specified number of peers from specified zone.
		@param int $numPeers - Number of peers to add.
		@param int $zone - Zone peers should exist in.
		@param string $fileID - FileID hash.
	*/
	function addPeer ($numPeers, $zone, $fileID){
		//get filename, size for file_id
		mysql_select_db("rivet");
		$sql = "SELECT filename,size FROM namemap WHERE `info_hash` = '$fileID'";
		$query = mysql_query($sql) or die(mysql_error());
		$row = mysql_fetch_row($query);
		$filename = $row[0];
		$size = ceil($row[1] / 1048576); //convert bytes to megabytes and rounds up

		//get ip with highest space
		mysql_select_db("master_db");
		$sql = "SELECT ip FROM peers WHERE zone = '$zone' ORDER BY space DESC"; //select IP
w highest space
		$query = mysql_query($sql) or die(mysql_error());

		/*
			1. Check to ensure peer doesn't already have file.
			2. If peer does have file, goto next peer.
			3. Else Add to current peer.
```

```
                4. Once enough peers have been added, break out of loop.
        */
        while($row = mysql_fetch_row($query)){
                //get out of here if enough peers have been added
                if ($numPeers == 0)
                        break;

                $hyphenIP = str_replace(".","-",$row[0]); //this is the table name for ip

                //see if the IP already has that file
                $sql = "SELECT filename FROM `$hyphenIP` WHERE `file_id` = '$fileID'";
                $exists = mysql_query($sql) or die(mysql_error());

                if(mysql_num_rows($exists) == 0){
                        //insert file into IP address table
                        $sql = "INSERT into `$hyphenIP` (filename,file_id) VALUES
('$filename','$fileID')\n";
                        mysql_query($sql) or die(mysql_error());

                        //update peer space left in peers table
                        $sql = "UPDATE peers SET space = space - $size WHERE ip = '$row[0]'";
                        mysql_query($sql) or die(mysql_error());

                        //call rss_create.php to update the peer
                        exec("php rss_create.php $row[0]");

                        //decrement number of peers left to add
                        $numPeers--;
                }

        }
}


//Get all IPs currently serving file <file_id>
$fileID = $argv[1];
$rivetID = "x".$argv[1]; //rivet stores files as x$fileID

//some vars to make things easier?
$db_user = 'root';
$db_password = 'netsys';
$db_address = 'localhost';

//connect to the server
mysql_connect ($db_address, $db_user, $db_password);
mysql_select_db('rivet');

//pull all ips for <file_id>
```

```
	$sql = "SELECT ip FROM `".$rivetID."`";
	$query = mysql_query ($sql) or die(mysql_error());

	//calculate how many seeds are in each zone
	$zonetotal = array();
	mysql_select_db('master_db'); //switch to master db

	while ($row = mysql_fetch_row($query)){
		//determine zone for each ip
		$sql = "SELECT zone FROM peers WHERE ip = '".$row[0]."'";
		$result = mysql_query ($sql) or die(mysql_error());
		$zone = mysql_fetch_row($result);

		//total zones
		$zonetotal[$zone[0]]++;
	}

	//ensure enough zones
	$sql = "SELECT zones FROM files WHERE file_id = '".$fileID."'";
	$query = mysql_query($sql) or die(mysql_error());
	$row = mysql_fetch_row($query);

	if ($row[0] > count($zonetotal)){
		//too few

		//figure out potential zones
		//populate potentialZone[0]=1, potentialZone[1]=2, .. for potentialZone[n]=n+1 where n
= total number of zones
		$potentialZones = array();//potential zones
		for ($i=0;$i<$numZones;$i++){
			$potentialZones[$i]=$i+1;
		}

		//loop through currently populated zones
		foreach ($zonetotal as $zone => $total){
			//for every potential zone check to see if seeds already exist in this zone
			foreach($potentialZones as $index => $zoneNum){
				//if they do delete the zone from potential zone list
				if($zoneNum == $zone){
					unset($potentialZones[$index]);
				}
			}
		}

		//how many zones to add
		$additionalZones = $row[0] - count($zonetotal);

		//randomly pick additional zones
```

```
                shuffle($potentialZones);
                $ind = 0;
                $addtoZones = array();
                while ($additionalZones > 0 ){
                        $addtoZones[$ind] = $potentialZones[$ind];
                        $ind++;
                        $additionalZones--;
                }

                //find seeds/zone value
                $sql = "SELECT seeds FROM files WHERE `file_id` = '$fileID'";
                $squery = mysql_query($sql) or die(mysql_error());
                $seeds = mysql_fetch_row($squery);

                //add to peers
                foreach ( $addtoZones as $index => $zone ){
                        //echo "zone: $zone | seeds: $seeds[0] -addpeer:zones\n";
                        addPeer($seeds[0],$zone,$fileID);
                }

        }

        //ensure enough seeds in current zones using $zonetotal

        //find seeds/zone value
        $sql = "SELECT seeds FROM files WHERE `file_id` = '$fileID'";
        $squery = mysql_query($sql) or die(mysql_error());
        $seeds = mysql_fetch_row($squery);

        foreach ($zonetotal as $zone => $total){
                //if not enough seeds
                if ($total < $seeds[0]){
                        $toAdd = $seeds[0] - $total; //subtract current number of seeds
                        addPeer($toAdd,$zone,$fileID);
                }
        }
?>
```

I

Works Cited

1. Boian, Florian M., and Rares F. Boian. "Solving Storage Limitation Using A Peer-to-Peer Web File System." *IEEE Xplore*. Bolyai University, 2008. Web. 16 Apr. 2010.

2. Butt, Ali R., Troy A. Johnson, Yili Zheng, and Y. C. Hu. "Kosha: A Peer-to-Peer Enhancment for the Network File System." *ACM*. Purdue University, 6 Nov. 2004. Web. 16 Apr. 2010.

3. Jarraya, Houssem, and Maryline Laurent-Maknavicius. "A Secure Peer-to-Peer Backup System." *ACM*. SudParis, 23 June 2007. Web. 16 Apr. 2010.

4. Landers, Martin, Han Zhang, and Kian-Lee Tan. "PeerStore: Better Performance in Peer-to-Peer Backup." *IEEE Xplore*. National University of Singapore, 2004. Web. 16 Apr. 2010.

5. Stein, C. A., Michael J. Tucker, and Margo I. Seltzer. "Building a Reliable Mutable File System on Peer-to-peer Storage." *IEEE Xplore*. Harvard University, 2002. Web. 16 Apr. 2010.