CSE 534: Fundamentals of Computer Networks Final Report

Anish Ahmed
Stony Brook University
anish.ahmed@stonybrook.edu

Bhushan Shah
Stony Brook University
bhushan.shah@stonybrook.edu

Abstract

Peer to peer networks constitute a high percentage of network traffic. Traditionally, p2p networks communicate over a TCP connection. This means there should be a significant data transfer overhead because of TCP interactions and also because of peer interactions. With this study we aim to identify how communication and data transfer policies of different popular p2p clients effect their efficiency. We identify a set metrics to judge the efficiency of p2p clients and run tests to evaluate them.

Repository: https://github.com/anish19/FCNProjectFiles/.

1 Introduction

Traditionally, most data transfer over networks was done over a centralized server-client model. However, increasing bandwidth, proliferation of residential computers with increasing capabilities, and digitization of media led to the advent of peer-to-peer file sharing systems, which facilitated reliable low-cost transfer of digital media directly between users. With time, many peer-to-peer file sharing protocols, such as BitTorrent, Gnutella, eDonkey etc., emerged.

Through this project, we attempt to analyze the performance of a few popular peer-to-peer applications, and try to interpret the results in terms of their underlying protocols.

2 Problem Description

Today, most of the data transfer in peer-to-peer file sharing systems takes place over TCP, involving TCP connections between peers themselves. The overall performance of such a system, in terms of efficiency, amount of data transfer, bandwidth usage etc., is impacted by the specific algorithms and policies implemented by the system. We try to quantify and measure the impact due to protocol specific policies and algorithms on the performance, for a few different peer-to-peer file sharing systems.

To measure the impact as described above, we use the following metrics.

1. **Extra bytes downloaded-** Extra downloaded bytes are the bytes that a specific client downloads over the total file size. For eg. If the total size of a file is 100 MB but total data downloaded by the client is 110 MB then extra bytes downloaded are 10 MB. The more the extra downloaded bytes, the lesser the efficiency of the protocol.
2. **Bytes downloaded per peer-** indicating whether some peers are prioritized over others. If a prioritization policy is in place then we will evaluate its effect on the efficiency of file sharing.
3. **File popularity-** Some protocols/applications are more suited for popular files while some for old files. We will measure the outcome of the policy selection(within clients) for different file types.
4. **Average download time-** Average download time each client takes for a same sized file. This is not a strong measurement criteria because avg. download time can also be affected by network traffic, host activity etc. Regardless we will measure this to obtain a reasonable result.
5. **Average download speed-** Average download speed for each client for a same file size.

This is not a strong measurement criteria because avg. download speed can also be affected by network traffic, host activity etc. Regardless we will measure this to obtain a reasonable result.

To measure the metrics listed above, we use a combination of the application GUI itself, as well as programs developed by us, as explained in the next section. We then attempt to interpret the results of our analysis in light of specific information we have about the respective systems.

3 Solution Methodology

To perform measurements of the aforementoned metrics, we have developed the following programs.

3.1 Tools

1. **Raw Packet Capture Utility (RPCU)** The Raw Packet Capture Utility has been develped in C using *tcpdump* and *libpcap*[1]. It captures all the packets passing through the NIC and filters them based on arguments provided at the command line. The command to run this utility is

   capture [-i interface] [-s string] BPFexpression
   i : The interface from which we want to capture packets
   s : The string we want to match in the payload
   BPFexpression : Expression to specify the BPF filters

   Specifying the interface is required because the internet connection will not necessarily be on the default interface. This gives the tester a freedom to specify one specific interface on which he/she wants to capture packets.

   The string matching add-on is used for displaying packet content for only those packets that have the specified substring in them. This helps in visually confirming the correctness of the p2p payload content.

   BPF[2] supports filtering packets, allowing a userspace process to supply a filter program that specifies which packets it wants to receive. For example, a *tcpdump* process may only want to receive packets that initiate a TCP connection. BPF only returns packets that pass the filter that the process supplies. This avoids copying unwanted packets from the operating system kernel to the process, greatly improving performance.

   The packet capture output of RPCU resembles the output of linux system call *tcpdump*. The Raw Packet Capture Utility is used instead of *tcpdump* because the utility has been customized to suit our analysis of p2p traffic. The RPCU writes the output to STDOUT and also to a CSV file. The data written to the CSV file is analysed by the Peer Data Analyser(PDA), described later.

2. **Peer Data Analyzer (PDA)** This has been written in Python since it's easier to work with CSV files in python than in C. The PDA uses the data extracted by the RPCU and analyses to calculate the number of bytes received from one peer per p2p application connection. The PDA performs this by extracting the IP address and packet size from the file produced by RPCU.

   PDA sums up the data received from each peer and computes to total bytes received by each peer to peer application.

3. **P2P Applications' GUI** The calculations for average time and average speed for a download is collected manually from the respective p2p applications via the GUI. The readings provided by p2p application are assumed to be correct.

3.2 Model

The model used is shown in **Figure 1**.

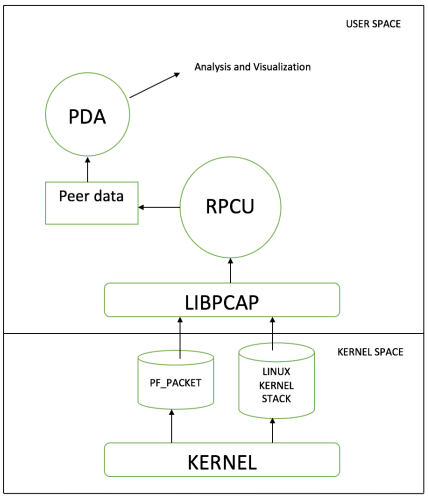

Figure 1: Model

In RPCU, libpcap is used to capture packets. The packets are filtered based on the p2p application being monitored. The filtered data is written out to standard output. Also, peer data is extracted from the packets and written to peer database in CSV format. These CSV files are used by Peer Data Analyser to perform per peer data transfer calculations. The information generated by PDA is finally used to analyse and visualize the performance of the respective p2p applications.

4 Evaluation and results

4.1 Evaluation

In order to evaluate, we found and downloaded ten files for each of the follwing p2p systems. We measured the aforementioned metrics using tcpdump, by looking at the traffic coming in through the specific ports used by these applications, and used the data we obtained to plot graphs. The code to do this can be found at `https://github.com/anish19/FCNProjectFiles/`.

- **BitTorrent**[3]
  The BitTorrent client is a very popular p2p application, with more than 150 million monthly users (as of 2012). It uses the BitTorrent protocol to collect file segments from other peers downloading the same file, as and when they become available. Peer discovery in Bittorrent protocol is performed using a Distributed Hash Table (DHT). The DHT is maintained at central servers (router.bittorrent.com and router.utorrent.com) to which connection is made at port 6881. The DHT only needs to provide address of one peer (minimum), after this the client can discover new peers from information exchanged with the discovered peers.[4]

- **Frostwire**[5]
  Frostwire is an open-source BitTorrent client developed as a fork of LimeWire (which was banned by US federal court in Oct 2010). It follows a query based model to collect segments of files. It was initially built on gnutella model[6]. On initial startup, the client software must bootstrap and find at least one other node. Various methods have been used for this, including a pre-existing address list of possibly working nodes shipped with the software, using updated web caches of known nodes (called Gnutella Web Caches), UDP host caches and, rarely, even IRC. Once connected, the client requests a list of working addresses. Then the client can perform searches on the connected nodes to discover content.

- **eMule**[7]
  eMule is used by users looking for rare content, and it uses a system of servers to store metadata about the files being shared. Peer discovery is done by querying the servers. The data transferred is compressed, and there are fast recovery mechanisms for corrupted downloads.

- **KickAssTorrents**[8]
  It is the most visited torrent directory in the world. It is built on bittorrent protocol but uses a special Magnet URI via cryptographic hash value to identify content on the network. The peer discovery scheme used here is simiiar to Bittorrent. The difference is in the algorithm used to identify the content at hosts. It used a cryptographic hash value rather than the location of the file to identify content.

Due to certain network restrictions, we had to connect to a VPN based in Amsterdam to be able to download files using these applications. Because of this, we were unable to get data about data being obtained from specific peers, and hence we could not measure the amount of data downloaded from each peer.

5 Results

Figure 2 shows the plot of percentage of extra bytes downloaded vs. file size for each download. Intuitively, we would expect the plot for a given p2p application to be monotonic.

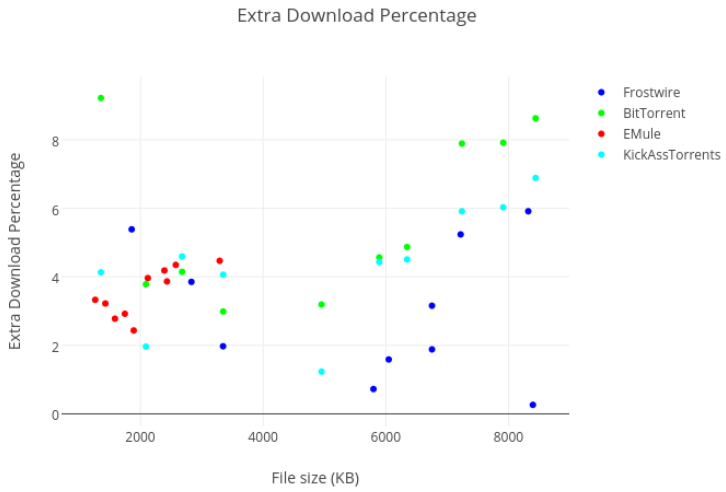

Figure 2: Extra download percentage

However, we see that the data points show a V-shaped behavior, with the extra download percentage first decreasing and then increasing after a point, as the file size increases.

The reason behind this is related to the information exchanged for peer discovery. The data exchanged for peer discovery constitutes (mainly, if not completely) the extra bytes downloaded. P2P applications discover new peers and send file segments simultaneously. For smaller file sizes, by the time a few peers are found the file is already downloaded by the initial peers. Therefore, the data exchanged for peer discovery is less (compared to larger files) because fewer peers are required to complete the download. So as the file size (for smaller files) increases, a small number of peers are sufficient to complete the download, hence the percentage of extra bytes downloaded decreases.

After a certain file size, the small number of peers are not sufficient to download the complete file before the peer discovery thread discovers new peers. This discovery of new peers requires more information exchange on the network. So as the file size increases the percentage of extra bytes required to discover new peers increases.

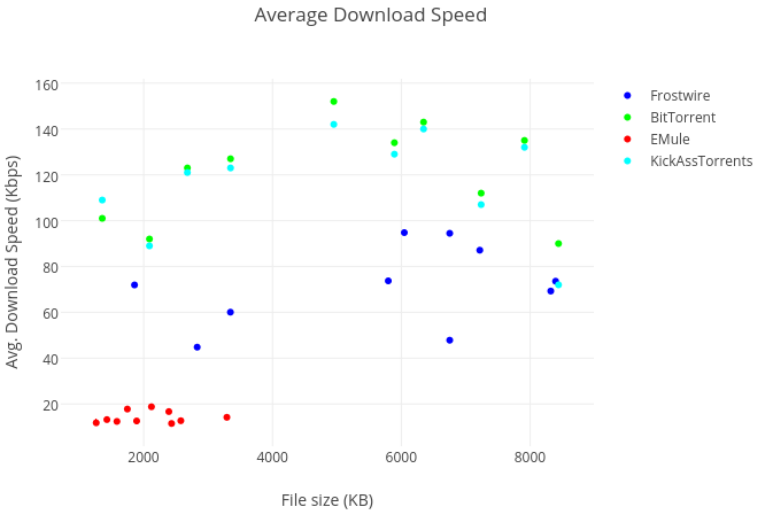

Figure 3: Average download speed

Figure 3 shows the plot of average download speed vs. file size for each download.

We see that for all the files downloaded using a specific p2p application, the average download speeds are more or less constant, and show no specific trends.

However, we see that across various p2p applications, we clearly obtain very different download speeds, depending on the protocols these applications use. For example, eMule offers very low download speeds, due to fewer peers, rarity of the files being downloaded, and a more complex server based model. BitTorrent and KickAssTorrents on the other hand offer higher download speeds due to high availability, as these are the two most popular p2p applications. Frostwire offers moderate download speeds.

We can conjecture that there is a direct relationship between the number of peers connected while downloading a file, and the average download speed with which the file gets downloaded. This is verified by the plot in **Figure 4**, which shows the same plot as in **Figure 3**, but with the data points resized according to the number of peers connected.

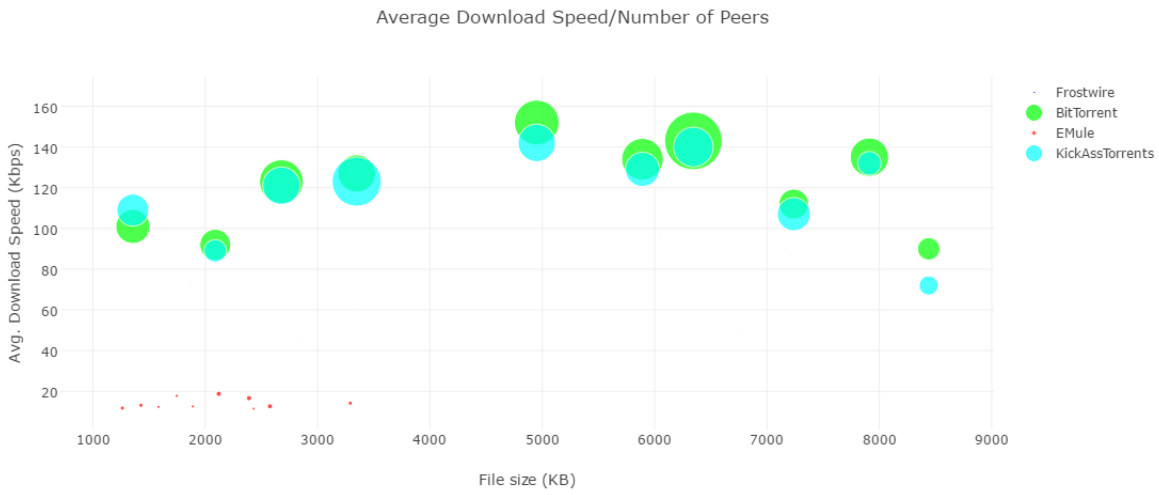

Figure 4: Relationship between average download speed and number of peers

We can see that BitTorrent and KickAssTorrents downloaded generally have a large number of peers connected, and also a higher download speeds. eMule downloads have fewer peers, and lower download speeds. Frostwire downloads have only one peer, but still offer higher download speeds than eMule downloads. This means that though the number of peers is a factor that determines the download speed, it is not the only factor.

6 Related work

1. *Peer-to-Peer Computing*[9]
   In this research the authors describe design and implementation of p2p networks. This detailed paper gives insight into various design decisions, algorithms and other characteristics.
2. *Peer-to-Peer Research at Stanford*[10]
   This paper describes latest research work in p2p applications at Stanford University. They describe various discovery and structuring techniques. They additionally cover the security aspect of p2p applications.
3. *Peer-to-Peer networks and computation*[11]
   The authors describe different models used in p2p systems. They cover different routing, searching and listing techniques used in the various models of p2p applications.

7 References

[1] tcpdump and libpcap -`http://www.tcpdump.org/`

[2] Berkeley Packet Filter - `https://en.wikipedia.org/wiki/Berkeley_Packet_Filter`

[3] BitTorrent - https://en.wikipedia.org/wiki/BitTorrent

[4] BitTorrent DHT - https://www.maketecheasier.com/how-bittorrent-dht-peer-discovery-works

[5] Frostwire - https://en.wikipedia.org/wiki/FrostWire

[6] Gnutella - https://en.wikipedia.org/wiki/Gnutella

[7] eMule - https://en.wikipedia.org/wiki/EMule

[8] KickAssTorrents - https://en.wikipedia.org/wiki/Magnet_URI_scheme

[9] Peer-to-Peer Computing - http://www.cs.uoregon.edu/Classes/F02.276/papers/p2p.pdf

[10] Peer-to-Peer Research at Stanford - http://www.cc.gatech.edu/~mihail/D.8802readings/garciamolina1.pdf

[11] Peer-to-Peer networks and computation - http://www.cai.sk/ojs/index.php/cai/article/viewFile/184/155