ENSC427
Communication Network

BitTorrent Protocol:
Priority Evaluation

Simon Fraser University

8888 University Drive
Burnaby, BC
V5A 1S6

Team 7

Charanpreet Parmar – csp6_at_sfu.ca

Feifan Jiang – feifanj_at_sfu.ca

Izaak Lee – igl_at_sfu.ca

http://www.sfu.ca/~csp6/

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Peer-to-peer file sharing is one of the biggest consumers of bandwidth in the Internet. BitTorrent is one of the most used file sharing protocols and thus, we will examine its effectiveness. Using ns2 as our simulator, we can examine the protocol in more depth. The key to the BitTorrent protocol operation is established via a TCP connection between the peers, and seeds in the swarm. The more clients in the swarm, the faster file transfer may occur. However, some users choose to limit their sharing ratio, hurting the health of the swarm. Using ns2 we will examine priority sharing in a swarm where good sharing ratio users get priority in downloading in hopes to allow the swarm as a whole to finish downloading sooner.

INTRODUCTION

Peer-to-peer (P2P) network is extremely common on the internet today. A specific protocol used for P2P networking is the BitTorrent. Up to 18% of internet traffic revolving around Peer-to-peer file sharing uses the BitTorrent protocol in North America.

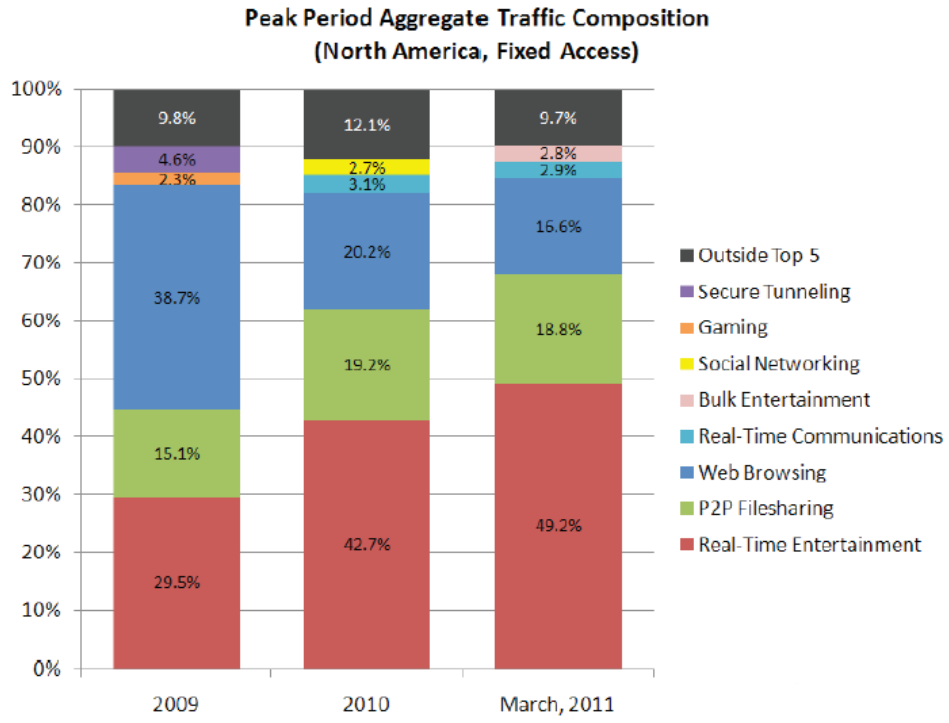

FIGURE 1 - TRAFFIC USAGE IN NORTH AMERICA OVER LAST 3 YEARS

Our report will break down the BitTorrent protocol and focus on interaction of the seeders and peers in the swarm. By gaining a deeper understanding of how peers share data with one another, we can implement a different protocol in hopes to improve the speed of the swarm. In particular we will be looking at the interaction between seeds and multiple peers, to see if prioritizing certain peers will improve the overall system.

MAIN SECTION

BITTORRENT

BitTorrent classifies its users with different names: seeds, peers and leechers, and the tracker. The two different users and the tracker combined form what BitTorrent refer to as the 'swarm'. Let us first understand the responsibility of each of these users.

SEEDER

A seeder is a user who has 100% of the data. A seeders job is to send out data to those who are missing it. If any users have collected 100% of the data, it will automatically become the seeder. The seeder will try its best to maximize the efficiency of the network by imploring certain protocols such as Rarest First Pieces of Super Seeding.

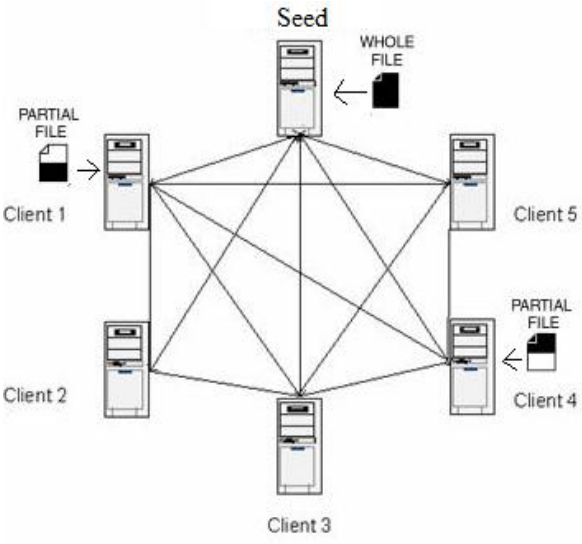

FIGURE 2- A FIGURE OF A SWARM WITH SEEDER WITH COMPLETE FILE

PEERS AND LEECHERS

Peers are the users who do not have 100% of the data. Peers will upload and download data. This is how a peer-to-peer network is created. Generally, the more peers in a network, the better, however, certain users are reluctant to share their data and only take from the network. These users are referred to as leechers. A high number of leechers in a swarm cause a negative effect.

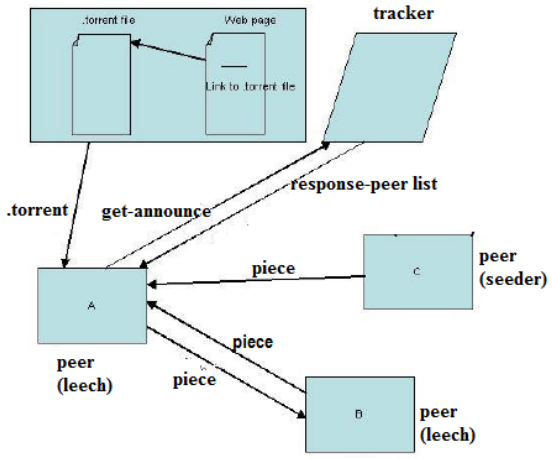

FIGURE 3- PEERS OBTAIN FILES AND SHARE FILE TO OTHER PEERS IN THE SWARM

TRACKER

The tracker is not a user. It is a server which keeps track of all the seeders and peers in the swarm, like an address book. Peers report to the tracker periodically to receive information from other peers and seeders. The tracker is no involved in the transferring of data directly; it does not have any data.

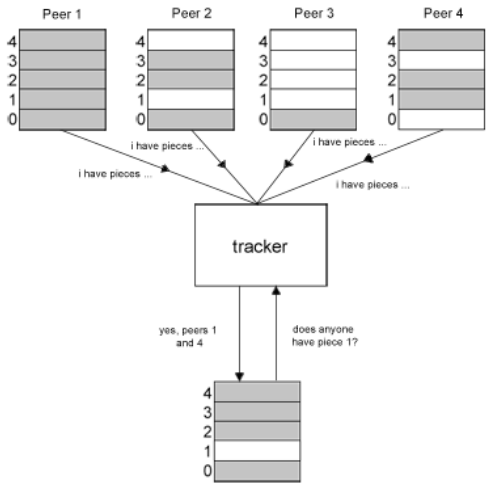

FIGURE 4 - TRACKER ACTS LIKE A CIRCUIT SWITCH BETWEEN PEERS

FILE TRANSFER

BitTorrent does not transfer files as a whole, but in parts. These parts are referred to as “pieces”. A file is broken into many pieces and is transferred from amongst peers. By using this technique, BitTorrent is extremely efficient at transferring large files. The amount of pieces a file is broken up to depends on the creator. To begin, a user creates a file called a .torrent file and places this file on the server. This file contains numerous amounts of information about the file.

BITTORRENT FILE STRUCTURE

When a peer requests for the file, only part of it is sent at a time. The created .torrent file contains information about the parts and size of each piece. The .torrent files stores metadata with specific parameters:

TABLE 1- PARAMATER IN A .TORRENT FILE

Parameters	**Definition**
Announce	Determines the URL of the tracker
Info	Maps the files to a directory specified by user so that BitTorrent knows where to send data to and from
Name	Directory where the file is saved
Piece length	Number of bytes per piece
Pieces	A hash list
Length	Size of entire file (bytes)
Files	Request data

The selection on the number of pieces and piece length is important. These parameters can result in faster or slower downloads for other peers. A user who creates this .torrent file, will become the first seeder of the network.

PROTOCOL

Now that we have defined and understood the filing system BitTorrent implements, we can begin to understand the protocol. To begin, a seeder creates a .torrent file and places the file on the World Wide Web or a server. A client seeking to obtain a file finds the torrent file on the server. The client who is interested in the file will download the .torrent file and connect to the server and register itself with the tracker.

The peer begins by asking for the first piece of data and the tracker will redirect this message to an available user in the network. In this case a seeder will respond and begin data transferring. If more peers enter the network, the seeder may start uploading to others and the peer may do the same thing. At any moment, if any of the peers obtain 100% of the file, they become seeders and continue to help the swarm, assuming they are not leechers as leecher usually leave the swarm immediately after it has obtained the file.

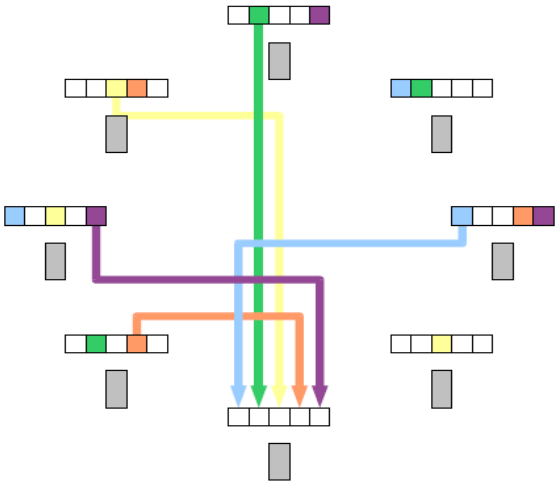

FIGURE 5- GRAPHICAL DEMONSTRATION OF P2P FILE SHARING

Once a peer completes the download of the piece and checks that the hash table matches, it will announce to the swarm that it has obtained the piece. The user will then begin to upload the piece to other users who need it. The peers will continue repeating this process and obtaining pieces needed to complete the file.

TIT-FOR-TAT STRATEGY

We note that the peers will automatically begin to upload to other peers. If they do not, then other clients will refuse to let him download. This is known as the Tit-for-Tat Strategy. Tit-for-Tat is one of the important principles in BitTorrent Protocol. The strategy is implemented through choking algorithm. The basic idea is to have peers upload pieces that another doesn't have thus exchanging these types of pieces – hence the name tit-for-tat. The method can enable the peers who have higher upload speed have higher likelihood for faster download time. Thus, the Tit-for-Tat strategy aims to improve the efficiency of distributing data.

DISCUSSION

We will be using ns-2.29 for our simulations. A BitTorrent like simulation is written by Kolja Edger. He quotes "BitTorrent like" because it does not completely implements a specific version of BitTorrent. This code aims at assessing the difference between a full simulation of network layers and simplified simulation on the application layer. Thus some functionality was simplified and others were not implemented.

When looking at the BitTorrent protocol, we can see that seeders choose to upload to peers who have high download speeds. By doing this, the protocol ensures helps ensure at least one file is floating in the network. In addition, by uploading to peers with higher download, those peers can help seed. However, most users leech and do not remain in the network after completion. When this occurs, it will slow down the download speed of other users who upload more.

To combat this, we look at the possibilities of having the seeder upload to those who upload fastest as opposed to those who download fast. The higher the upload speed, the faster other peers can receive pieces of the file. Therefore, we wish for these clients to receive files faster. If they become the seeders, the entire swarm benefits from their high sharing ratio.

PROBLEM: TRACKER

The easiest way to accomplish this is by looking at the tracker. The tracker will have all the information of all peers and seeds. However, the ns2 simulation is missing some functionalities and one such function is the tracker. Instead, the tracker is implemented in each node. So there is no stand-alone entity.

Because of this implementation by Kolja Edger, the changes needed to be made became difficult. Instead, we simulated a situation where those who upload faster start in the network earlier. By doing this, those who upload quickly will have the benefit of getting files first. Since the high uploaders receive the file first, the new peers entering the swarm will benefit from their high upload rate. Overall, by doing this we should see an improvement on the overall swarms download time.

In reality, this sort of implementation would involve the tracker having a long term average of each peers upload and downloads rate and would mean having a more calculation heavy tracker. Realistically, this would most likely only be possible on smaller private trackers rather than on larger public ones due to added complexity. Many private trackers already having similar systems in place where different user classes, typically divided based on their sharing ratio, get access to torrents after a certain time period after their initial release.

IMPLEMENTATION

What we are interested in seeing is the overall quality of the swarm. We want to observe the time it takes all peers to download a single file. One swarm will have the original coding where peers with highest download rate get priority from the seeder, and on the other hand the other swarm will use the code we edited.

In our code, we will implement a timing which determines which peer enters the network first. The peer which enters the network first will be those who have highest upload rates. This is to simulate the priority given to peers with higher upload rates. The next fastest peers will have priority and will be enter the swarm at an even interval of 1, 5, or 10 seconds. This will continue until all the peers have entered the network.

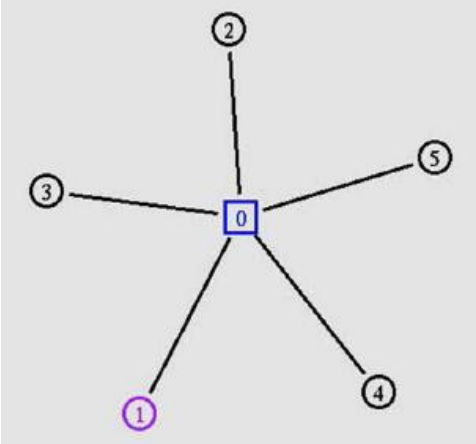

FIGURE 6 - A SIMULATION IN NS2 OF 1 SEED, 4 PEERS

By doing this, those who have highest upload rates will be allowed in first and they will download first.

DATA

We first begin with a base line test. This test will be simulated with a client size of 10. In the swarm, there will be 5 seeders and 10 peers. The seeders will be uploading at a rate of 0.5 MB/s. One of the peers that is downloading will be uploading at 1.0 MB/s and while others are at 0.5 MB/s. In addition, the download rate of all peers is set constant at 1.0 MB/s.

TABLE 2 - SEEDER AND PEER PARAMETERS

	Upload Speed (MB/s)	Download Speed (MB/s)
Seeder	0.5	N/A
Peers	1.0 or 0.5	1.0

We begin the first simulation and allow all the peers to join at once. With this, the peers who have the highest download rate will have priority and begin downloading first. With this, we have the log file.

TABLE 3 - LOG STATISTICS OF 5 SEED, 10 PEERS. NO PRIORITY

ID	Start_time	first_chunk_time	download_finished	stop_time	download_finished - start time
1	0	0	-1	118.708	-1
2	0	0	-1	118.708	-1
3	0	0	-1	118.708	-1
4	0	0	-1	118.708	-1
5	0	0	-1	118.708	-1
6	0	5.93705	114.619	118.708	114.619
7	0	15.6919	118.071	118.708	118.071
8	0	26.4805	115.687	118.708	115.687
9	0	23.8726	118.708	118.708	118.708
10	0	25.4215	116.26	118.708	116.26

From the table, we can see that the start time of all peers are the same. The script ignores who has the fastest uploading speeds. As a result, the total time it took for all the peers to attain the file took a total of 118.708 seconds simulation time.

Now we can look at the upload priority. The script is set such that those who have the highest uploaded speed enter the network first. By doing this, that specific peer will be downloading first regardless of its download rate. We can see this in the table below.

TABLE 4 - LOG STATSTICS: 5 SEEDS, 10 PEERS. 1 SECOND PRIORITY

ID	Start_time	first_chunk_time	download_finished	stop_time	download_finished - start time
1	0	0	-1	112.33	-1
2	0	0	-1	112.33	-1
3	0	0	-1	112.33	-1
4	0	0	-1	112.33	-1
5	0	0	-1	112.33	-1
6	1	13.2152	112.33	112.33	111.33
7	1	7.93149	111.994	112.33	110.994
8	0	3.69505	110.626	112.33	110.626
9	1	24.7416	111.535	112.33	110.535
10	1	17.4676	111.307	112.33	110.307

In this case, the other peers start 1 second later than peer 8. With that, the total time it took for every peer to get the file decreased. If we graph the flow level of the packets, we can examine the amount of data being transferred and between nodes and seeds. We will first examine node 8 which is the first to receive pieces from seeds.

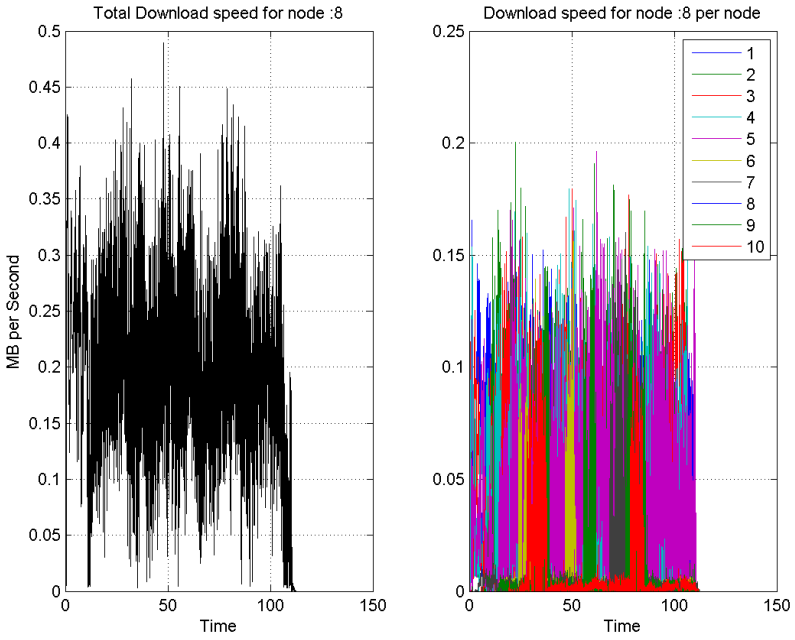

FIGURE 7 - NODE 8 - LEFT: TOTALY DOWNLOAD SPEED. RIGHT: UPLOAD SPEED FROM OTHER NODES.

We can see from this graph, that the total download speed for node 8 is immediate from time 0. Next we look at the upload of node 8.

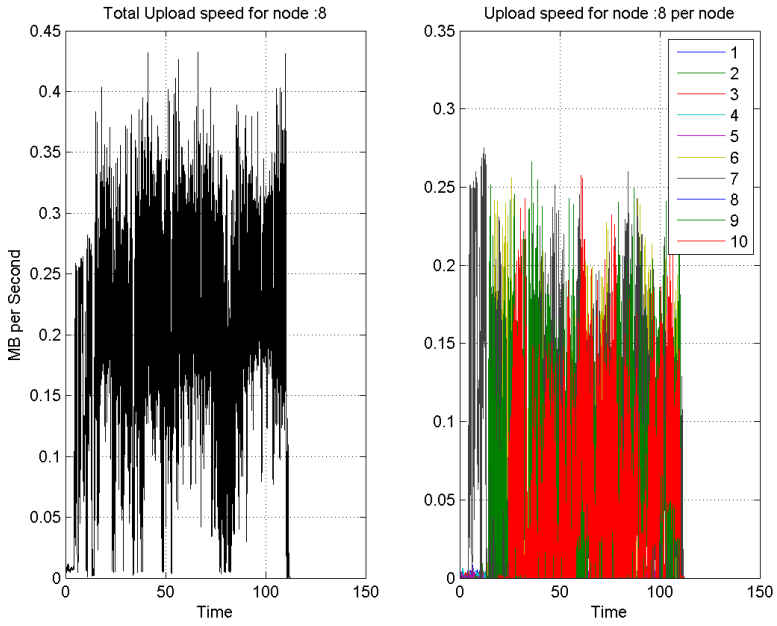

FIGURE 8 - NODE 8 - LEFT: TOP UPLOADING SPEED. RIGHT: AMOUNT UPLOADED TO INDIVIDUAL NODES.

We can see that the peer does not begin to upload until time = 1. This is correct since node 8 has no peers to share with until time =1. We continue testing with other start times and yield the following results.

TABLE 5- LOG STATISTICS: 5 SEEDS, 10 PEERS. 5 SECOND PRIORITY.

ID	Start_time	first_chunk_time	download_finished	stop_time	download_finished - start time
1	0	0	-1	111.765	-1
2	0	0	-1	111.765	-1
3	0	0	-1	111.765	-1
4	0	0	-1	111.765	-1
5	0	0	-1	111.765	-1
6	5	10.8781	108.697	111.765	103.697
7	5	21.4159	110.345	111.765	105.345
8	0	6.88248	105.088	111.765	105.088
9	5	24.8195	111.765	111.765	106.765
10	5	20.5582	109.527	111.765	104.527

Again with this, we can see that the overall system improved. The graphs are relatively the same as the previous simulation except the peers join the swarm at timer = 5 seconds. This resulted in a slight improvement once again.

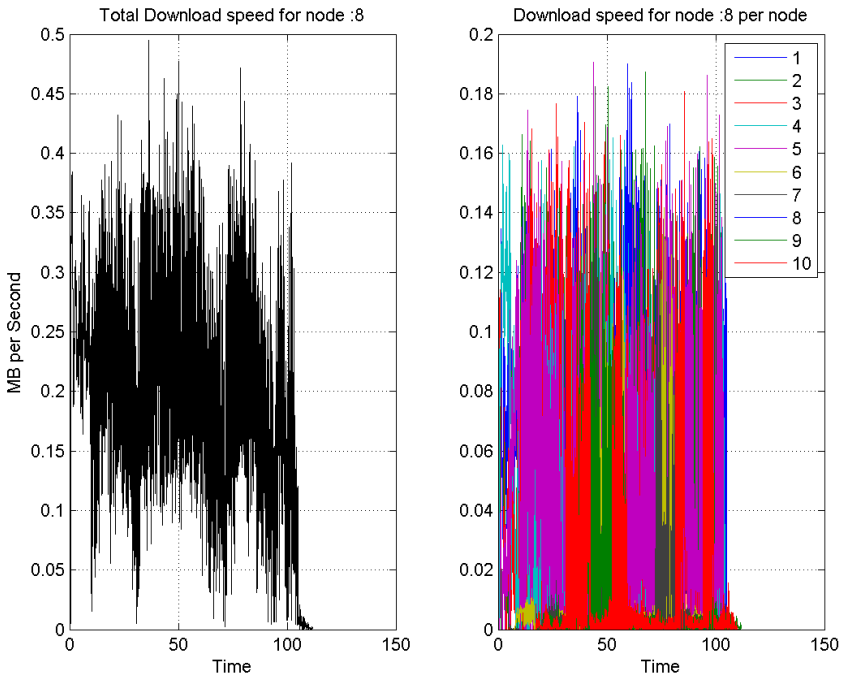

FIGURE 9 – NODE 8 – LEFT: TOTAL DOWNLOAD SPEED. RIGHT: DOWNLOAD SPEED FROM OTHER NODES

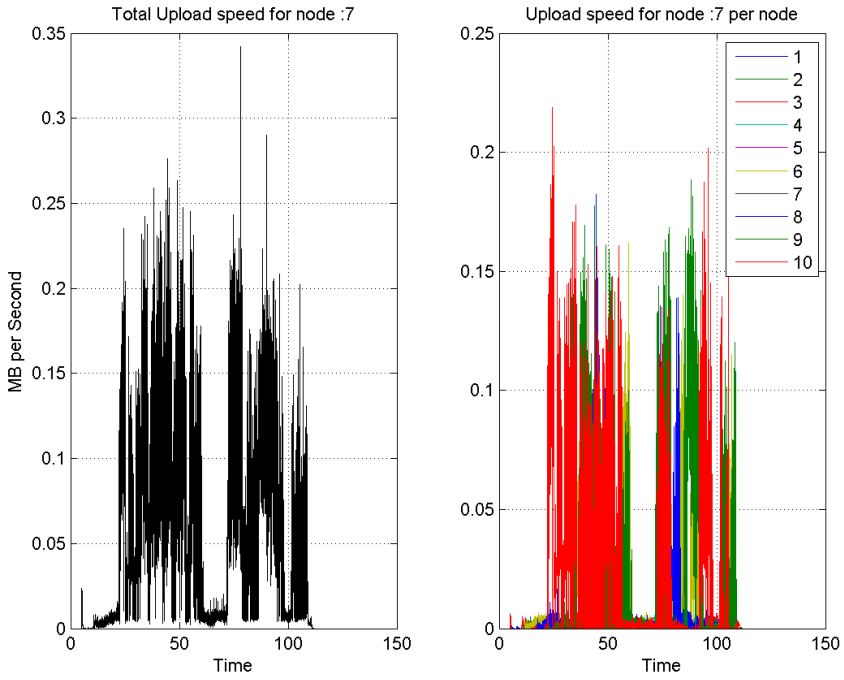

FIGURE 10 – NODE 8 – LEFT: TOP UPLOADING SPEED. RIGHT: AMOUNT UPLOADED TO INDIVIDUAL NODES

Lastly, we test the system with a 10 second delay for other peers.

TABLE 6 – LOG STATISTICS: 5 SEEDS, 10 PEERS. 10 SECOND PRIORITY

ID	Start_time	first_chunk_time	download_finished	stop_time	download_finished - start time
1	0	0	-1	118.203	-1
2	0	0	-1	118.203	-1
3	0	0	-1	118.203	-1
4	0	0	-1	118.203	-1
5	0	0	-1	118.203	-1
6	10	15.5714	116.189	118.203	106.189
7	10	19.7144	118.203	118.203	108.203
8	0	6.90978	116.741	118.203	116.741
9	10	14.3465	117.786	118.203	107.786
10	10	28.8848	115.676	118.203	105.676

In this simulation, this total time is only slightly below the base line test. This could be the result of the other peers joining too late. The seeds spend a lot of time on the single node in the swarm, node 8 can only upload 1mb/s and no more. If the seeds spend too long on a single peer then they waste their multitasking capabilities. Node 8 can only upload 1mb/s and no more.

CONCLUSION

From the simulations, we can see that a swarm can benefit from using upload priority instead of download priority. However, we notice that if seeds spend too much time on small number of individuals, it can have a reverse effect on the swarm.

FUTURE WORK

From our work, we can see that there is a slight improvement in the overall network. However, our implementation is a static network where no one leaves or enters after the nodes are all started, as in there are no leechers, the tracker already has knowledge of each peers potential bandwidth. This clearly would never occur in a real network. In order for us to further investigate this priority protocol, we need a complete implementation of the BitTorrent protocol on ns2. A complete BitTorrent protocol will include the tracker which will contain information about peers. This will significantly help in terms of priority implementation.

Currently, our priority implementation allows peers who have high upload speeds to enter the network first. However, once other peers join, the seeds give them priority because of their download speeds. Our algorithm forces the seeders to only upload to the good peers. In the future, seeders need to give peers with greater upload speed priority automatically. Until then, the system is not truly autonomous.

In addition, we have run a small sample size and yielded mostly positive result. However, we need to continue to scale upwards to more realistic numbers to obtain a better understanding of the swarm. With our results, we hope others, with greater coding ability than us, can look into upload priority in BitTorrent and further improve the algorithm.

WORKS CITED

[1] B. Cohen, "BitTorrent," 10 1 2008. [Online]. Available: http://www.bittorrent.org/beps/bep_0003.html. [Accessed 1 3 2012].

[2] D. Schoder, K. Fischbach and C. Schmitt, "Core Concepts in peer to peer networking," University Cologne, Germany, 2005.

[3] K. Aberer, "Distributed Data Management Peer-to-Peer System," 2006.

[4] R. Steunmetz and K. Wehrle, "Peer-to-peer Systems," 2005.

[5] J. E. Berkes, "Descentralized Peer-to-Peer Network ArchitectureL Gnutella and Freenet," Winnipeg, 2003.

[6] J. Chung and M. Claypool, "NS By Example," Worcester polytechic institue, [Online]. Available: nile.wpi.edu/NS/. [Accessed 03 04 2012].

[7] M. M. Sasan Hezarkhani, "Analysis of Live Video Streaming Over Bittorrent Peer-to-Peer Protocol," Simon Fraser University, Vancouver, 2011.

[8] "Liberty Voice," 26 october 2010. [Online]. Available: http://www.libertyvoice.net/2010-10/bittorrent-still-dominates-global-internet-traffic/.

[9] A. Leon-garcia and I. Widjaja, Communication Networks: Fundamental Concepts and Key Architecture, New York: New York: Elizabeth A. Johns, 2004.

[10] D. Erman, D. Ilie and A. Popescu, "BitTorrent Session Characteristcs and Models".

[11] L. A., "Rarest First and Choke Algorithm are Enough".

[12] A. R. Bharambe and C. Herley, "Analyzing and Improving BitTorrent Performance".

[13] M. Baker and R. Lakhoo, "Peer-to-Peer Simulator".

[14] E. Ayele, "Analysis and deployment of the BitTorrent protocol for community Ad-hol Network".

[15] K. Eger, "BitTorrent in ns-2," 11 January 2012. [Online]. Available: https://sites.google.com/site/koljaeger/bittorrent-simulation-in-ns-2. [Accessed 3 April 2012].

APPENDIX

DEFINITION

UPLOADING

defined as sending data from a local system to a remote system.

DOWNLOADING

defined as receiving data from a remote system.

HANDSHAKE

Handshake includes peer id and info field hash. During the handshake process, peers send the information of the pieces of data they are processing to each other. Peers send a 20 byte SHA1 hash of the encoded info value from the metainfo and a 20 byte peer id. If the peer id or harsh value does not match the one expected, the connection is close.

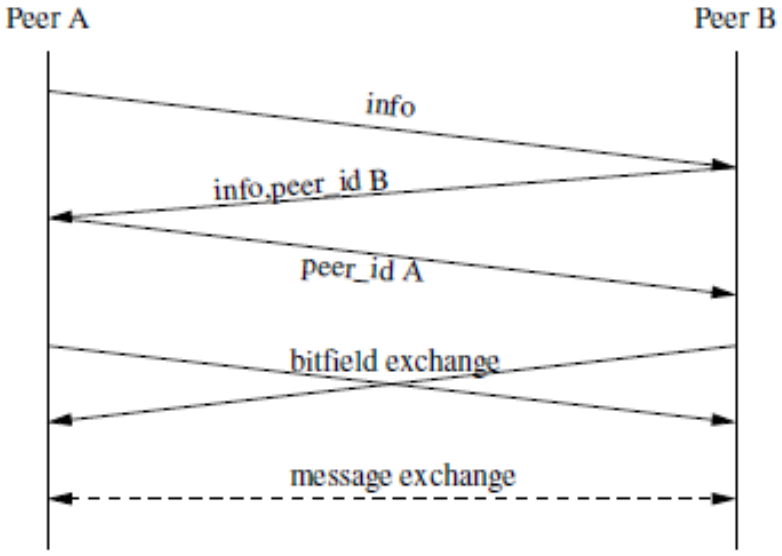

FIGURE 11 - DEMONSTRATION OF 3 WAY HANDSHAKING IN BITTORRENT

RANDOM FIRST PIECE AND RAREST FIRST PIECE

In BitTorrent Protocol, pieces of data are requested followed by the principles of random first piece and rarest first piece. In most case scenarios of Peer-to-Peer network, tracker will find the minimal occurrence piece and set that piece to a high value, which is rarest first piece. For the cases that rarest first piece doesn't fit, peer will request a random piece i.e. random first piece.

SUPER SEEDING

In the case of super seeding, a seeder masquerades itself as peer, and as peers enter the swarm. The masqueraded seed will announce to new peer it has a piece he is willing to give them. This piece of the file will be a piece which does not exist in the swarm. The seeder will continue to masquerade itself and continue delivering non-existing pieces until all pieces of the file exist in the swarm. Once this is accomplished, the seeder will act as a normal seeder once again. By doing this, it protects the swarm from dying.

CHOKING ALGORITHM/ INTERESTED MECHANISM

Choke/unchoke and interested/not interested are responsible for the fairness in BitTorrent Protocol. Choked peers are not allowed to download data from the one who enable the choke algorithm on peers. The choked peers will be enabled to upload and download until the peers go back to the unchoke state. Interested/not interested provide peers information about who are holding the peers' missing pieces of data.

The choking algorithm and interested mechanism cooperate to give peers certain priority that in general punishes the peers who share less resource. However, we cannot compare the algorithm benefits the peers share the most and the one share moderately. This consideration becomes our motivation to simulate 4 different tests to find out some interested data for the peers with outstanding upload speed.

CODE DESCRIPTION

FILES

A list of files use to create our simulation.

TABLE 7 - FILE DESCRIPTION

Name	Description
BitTorrent.tcl	Contains the number of seeds and peers to be generated. Sets the start time of peers, allowing those who upload fastest to have priority.
BitTorrent_app.cc	Contains the Constructor and destructor of nodes. Algorithm for handshakes, creating messages, choking, super seeding, receiving and sending messages.
BitTorrent_app_flowlevel.cc	Creates channels between peers and checks channel. Also responsible for checking choking and handle requests.
BitTorrent_connections.cc	Generates TCP agent and blinds to application node. Stores all connection ID and destination IDs.
BitTorrent_data.cc	Creates all packet and header lengths.
BitTorrent_tracker.cc	A simple tracker which keep tracks of file size and chunk size. Also responsible for deleting and adding peers to list.
BitTorrent_tracker_flowlevel.cc	Responsible for keeping track of rarest piece in network. Also keeps track of peer list.

FUNCTION DESCRIPTION

There are many functions in this program. We will be naming a few of the important ones.

TABLE 8 - FUNCTION DESCRIPTION

Function name	Description
Fully_meshed2	Links the nodes to with a TCP connection and sets them with a specific upload and download speed.
Priority_tracker	Determines and set the starting time of nodes depending on their upload speeds. This is used to set priority for those who have high upload speeds.
BitTorrentApp	Constructs the nodes to be either seeds or peers.

Tracker_request	Gets ID from tracker and checks connections.
get_ids_from_tracker	Send out list of id from tracker, if peer ID not in set, add.
Check_connections	Checks connections otherwise timeout.
make_new_peer_list_entry	Post all information about itself to tracker and adds it to the peer list.
Connect	Builds TCP agents on both the source side and destination side. Also calls upon connect_step_2 and connect_step_3
RECV	Handles the connection of agents and handles all situations of messages received.
Handle_recv_msg	If messages are at least one message size or greater, this function will handle the received message.
Check_interest	See which peers are interested.
Check_connections	Check connections between all peers.
Check_choking	Checks choking and updates the peer list.
Make_request	Function makes a request to peers and seeds for file.
Chunk_complete	Function disconnects peers from others and makes himself a seeder before rejoining the swarm.
Handle_handshake	This function will handle the 3 way handshake.
BitTorrentAppFlowlevel	Sets the download, chunk and request of the pieces needed.
BitTorrentConnection	Sets all the destination and source addresses between peers and is responsible for closing connections.
BitTorrentData	Determines the size of all data.
BitTorrentTracker	Contains the information about file size and piece size.
Reg_peer	Register peer for the tracker.
Del_peer	Deletes peer from the tracker.
Return_rarest_chunk	Returns the rarest piece existing in the network.

FLOW CHART: BITTORRENT UPPER LEVEL

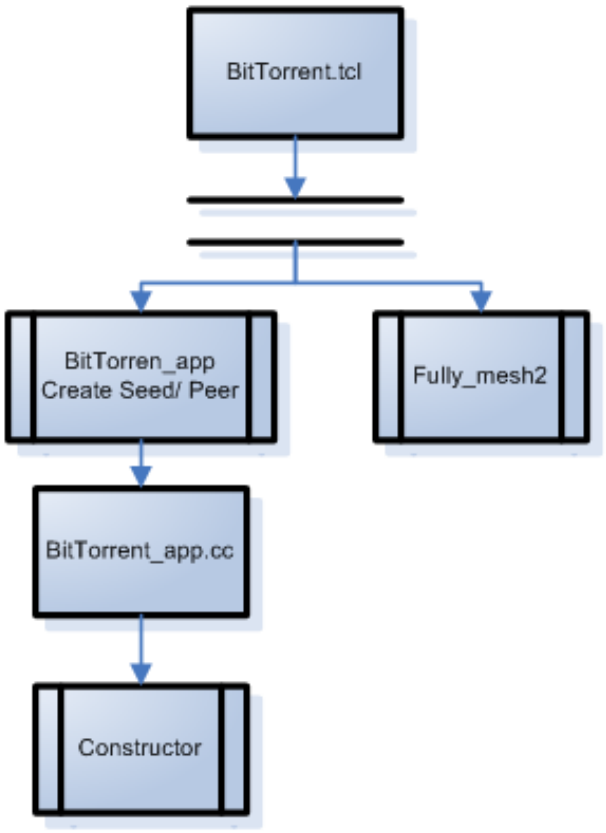

FIGURE 12 - THE UPPER LEVEL SCRIPT WHICH IS RESPONSIBLE FOR CALLING FUNCTIONS TO CREATE ALL THE NODES AND LINKS

FLOW CHART: RECV FUNCTION

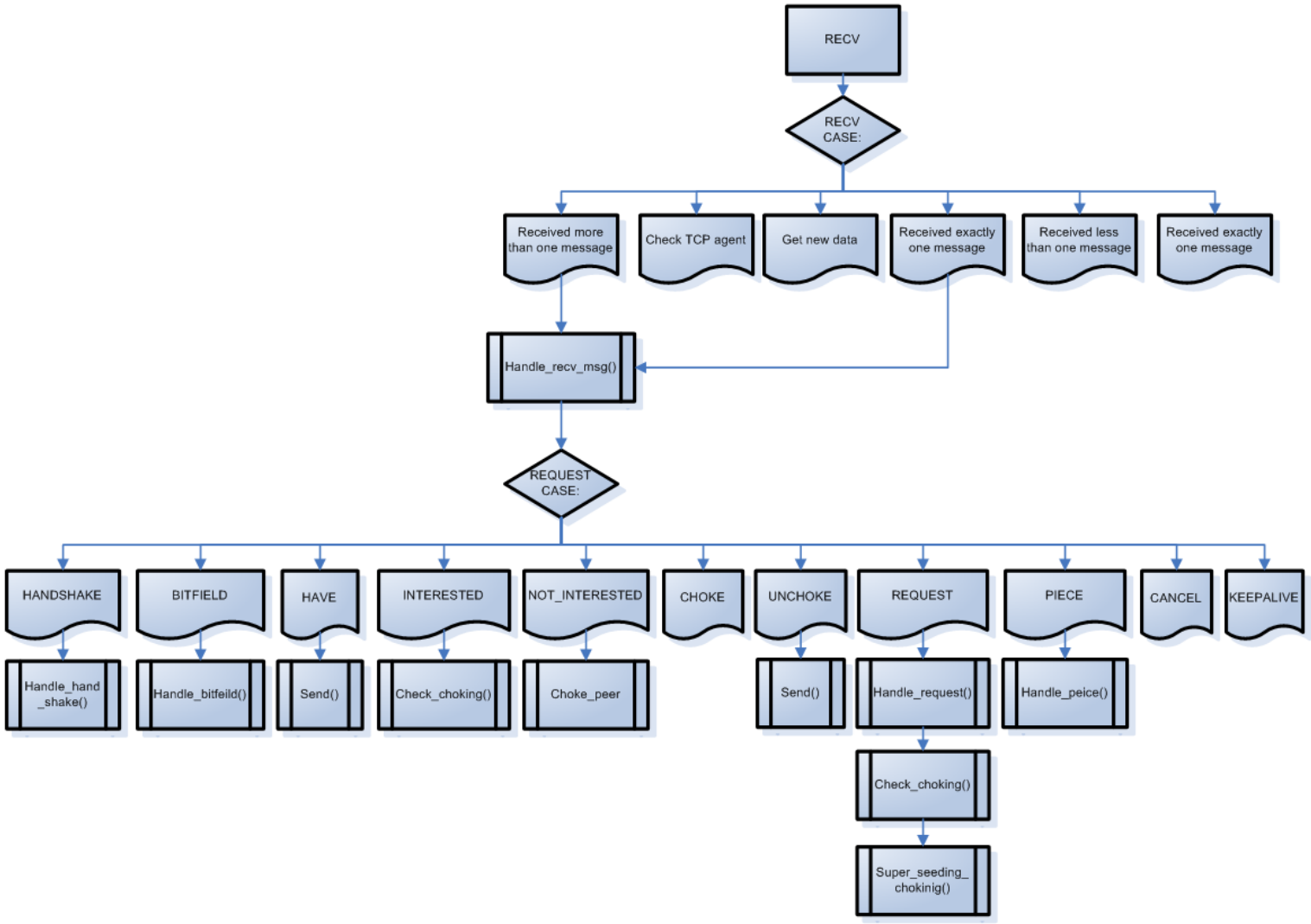

FIGURE 13 - RECV FUNCTION

Flow chart: Peers

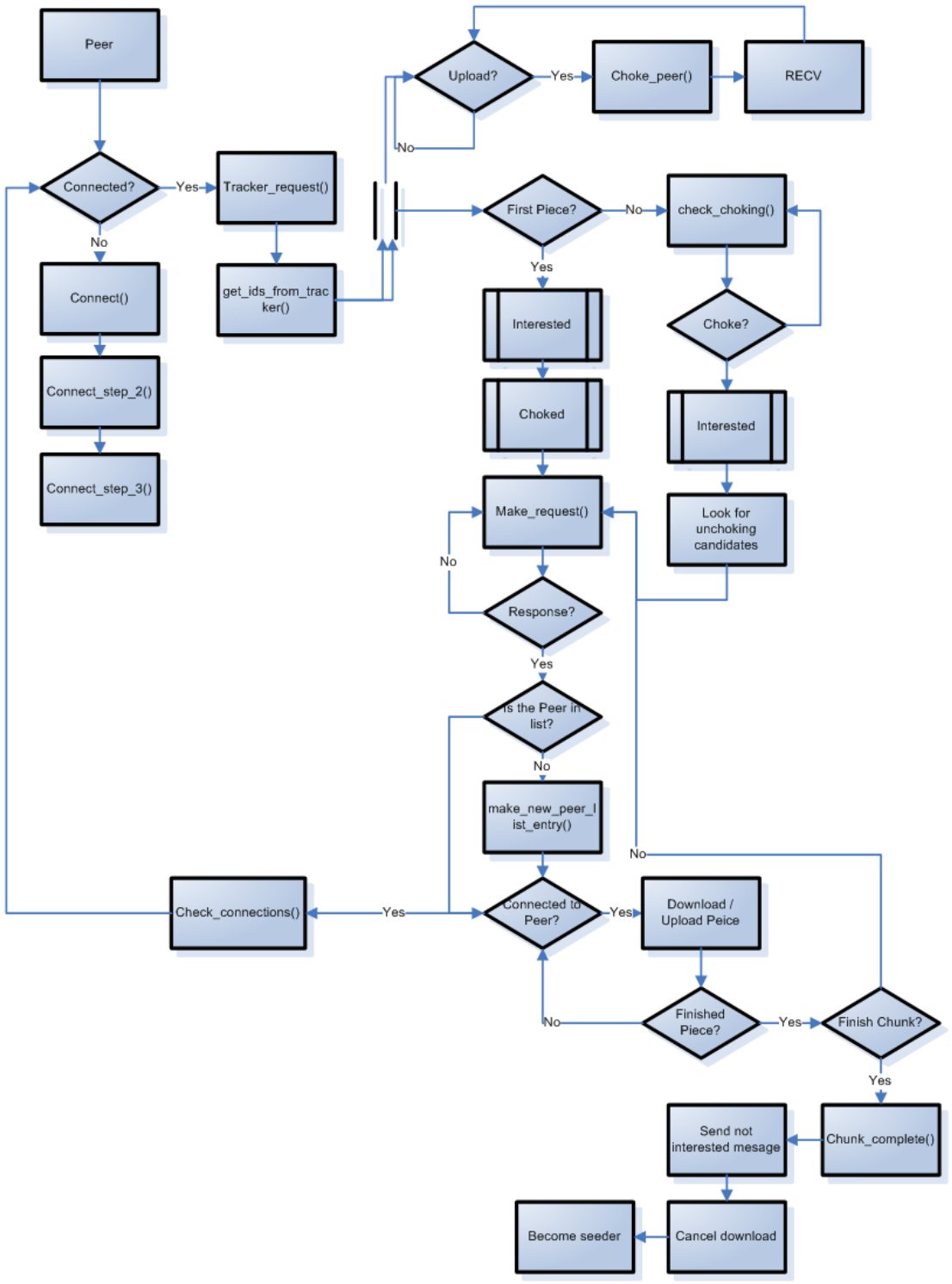

FIGURE 14 - PEERS

FLOW CHARTS: SEEDS

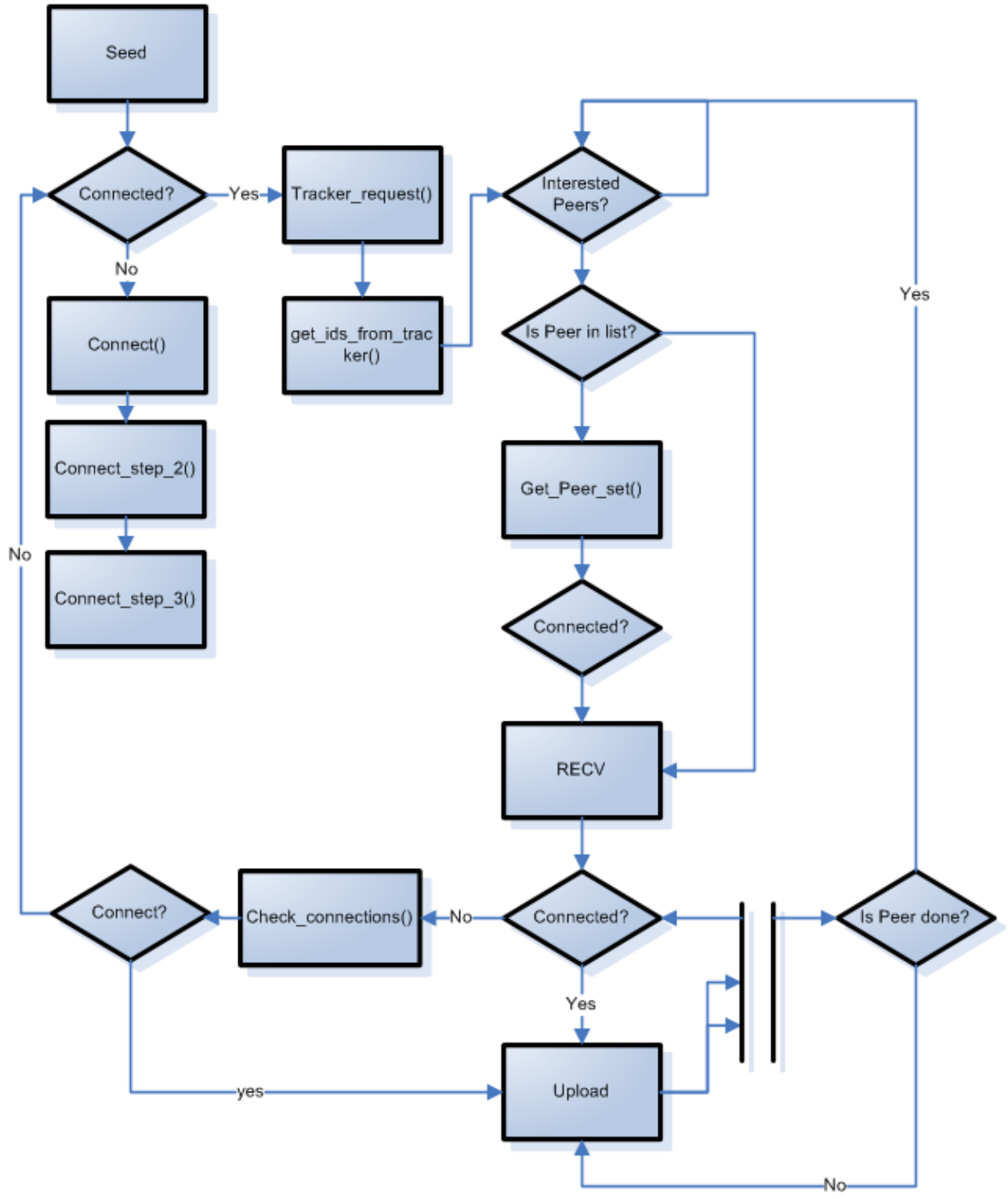

FIGURE 15 - SEEDS

CODE LISTING

data.m

```
%%Open
% Parameters
File = 'bt.tr';
samp_rate = 80;
num_peers = 10;
packets = 1;
upload_rate = 1;
download_rate = 1;
downloaded = 1;
save_fig = 1;
fig_type = 'fig';
% Open File
fID = fopen(File);
%% Reading all Data
indx = 1;
while(~feof(fID))
    event(indx) = {fscanf(fID, '%s', 1)};
    time(indx) = {fscanf(fID, '%s', 1)};
    node_from(indx) = {fscanf(fID, '%s', 1)};
    node_to(indx) = {fscanf(fID, '%s', 1)};
    packet_type(indx) = {fscanf(fID, '%s', 1)};
    packet_size(indx) = {fscanf(fID, '%s', 1)};
    flags(indx) = {fscanf(fID, '%s', 1)};
    fid(indx) = {fscanf(fID, '%s', 1)};
    src_addr(indx) = {fscanf(fID, '%s', 1)};
    dest_addr(indx) = {fscanf(fID, '%s', 1)};
    seq_num(indx) = {fscanf(fID, '%s', 1)};
    pkt_id(indx) = {fscanf(fID, '%s', 1)};
    indx = indx + 1;
end
%%
fclose(fID);
%%Extract Received bits only (ingnore queue and enqueue since they get
%%received eventually)
j = 1;
for i = 1:indx-1
    if cell2mat(event(i)) == 'r'
        time_r(j) = str2double(cell2mat(time(i)));

bits_r(j,floor(str2double(cell2mat(dest_addr(i)))),floor(str2double(cell2mat(
src_addr(i))))) = str2double(cell2mat(packet_size(i)));
        j = j + 1;
    end
end
clear indx
%%
%plot(time_r, bits_r(:,peer it's to, peer it's from))
%num_peers = 10;%max(floor(str2num(cell2mat(transpose(unique(dest_addr))))));
for i = 1:num_peers
    for j = 1:num_peers
        for k = 1:length(bits_r)-samp_rate-1
```

```
            bps(k:k+samp_rate-1,i,j) = (sum(bits_r(1:k+samp_rate-1,j,i)) -
sum(bits_r(1:k,j,i)))/(time_r(k+samp_rate)-time_r(k))/(1024*1024);
            u_bps(k:k+samp_rate-1,i,j) = ((sum(bits_r(1:k+samp_rate,i,j)) -
sum(bits_r(1:k,i,j)))/(time_r(k+samp_rate)-time_r(k)))/(1024*1024);
        end
    end
end
%%
num_plots = packets + upload_rate + download_rate + downloaded;
for i = 1:num_plots:num_plots*num_peers
    sp_x = ceil(sqrt(num_peers));
    sp_y = floor(sqrt(num_peers));
    while(sp_x*sp_y < num_peers)
        sp_y = sp_y + 1;
    end
    node_current = ceil(i/num_plots);
    if downloaded
        h = figure(i + downloaded - 1);
        for j = 1:num_peers
            subplot(sp_y,sp_x,j)
            plot(time_r, cumsum(bits_r(:, j, node_current)))
            title(strcat('To node :', num2str(j), ', From node :',
num2str(node_current)))
            xlabel('Time')
            ylabel('Number of bits')
            grid on
        end
        set(gcf, 'Position', get(0,'Screensize'));
        if save_fig
            saveas(h,strcat('Node',
num2str(node_current),'_downloaded'),fig_type);
        end
    end
    if upload_rate
        h = figure(i + upload_rate + downloaded - 1);
        subplot(1,2,1);
        plot(time_r(1:length(u_bps)),
sum(transpose(u_bps(:,1:end,node_current))),'-k');
        title(strcat('Total Upload speed for node :', num2str(node_current)))
        xlabel('Time')
        ylabel('MB per Second')
        grid on
        subplot(1,2,2);
        plot(time_r(1:length(u_bps)), u_bps(:,1:end,node_current));
        title(strcat('Upload speed for node :', num2str(node_current),' per
node'))
        xlabel('Time')
        grid on
        for k = 1:num_peers
            leg(k) = {num2str(k)};
        end
        legend(leg)
        set(gcf, 'Position', get(0,'Screensize'));
        if save_fig
            saveas(h,strcat('Node',
num2str(node_current),'_upload_rate'),fig_type);
        end
```

```
    end
    if packets
        h = figure(i + upload_rate + downloaded + packets - 1);
        for j = 1:num_peers
            subplot(sp_y,sp_x,j);
            plot(time_r, bits_r(:,j,node_current));
            title(strcat('To node :', num2str(j), ', From node :',
num2str(node_current)))
            xlabel('Time')
            ylabel('Packet size')
            grid on
        end
        set(gcf, 'Position', get(0,'Screensize'));
        if save_fig
            saveas(h,strcat('Node',
num2str(node_current),'_packets'),fig_type);
        end
    end
    if download_rate
        h = figure(i + upload_rate + downloaded + download_rate + packets -
1);
        subplot(1,2,1);
        plot(time_r(1:length(bps)),
sum(transpose(bps(:,1:end,node_current))),'-k');
        title(strcat('Total Download speed for node :',
num2str(node_current)))
        xlabel('Time')
        ylabel('MB per Second')
        grid on
        subplot(1,2,2);
        plot(time_r(1:length(bps)), bps(:,1:end,node_current));
        title(strcat('Download speed for node :', num2str(node_current),' per
node'))
        xlabel('Time')
        grid on
        for k = 1:num_peers
            leg(k) = {num2str(k)};
        end
        legend(leg)
        set(gcf, 'Position', get(0,'Screensize'));
        if save_fig
            saveas(h,strcat('Node',
num2str(node_current),'_download_rate'),fig_type);
        end
    end
end
```

bittorrent.tcl

```
#Total number of Peers
set no_of_peers 10
#Number of peers which are seeds
set no_of_seeds 5
#Seed for Random Number Generator
set s 1
#toggle for Seeder Priority
set priority 0
#time_delay
set time_delay 10.0

#Set Upload speed for all Peers
for {set i 0} {$i < $no_of_peers} {incr i} {
    set C_up($i) [expr 500*1024]
}
#Change Upload speed for select peers
set C_up(7) [expr 1000 * 1024]

#Create a simulator object
set ns [new Simulator]

remove-all-packet-headers
add-packet-header IP TCP Flags

#Use Heap Scheduler
$ns use-scheduler Heap

#set the routing protocol
$ns rtproto Manual

# Simulation Parameters:
source /home/charanpreet/ns-allinone-2.29/ns-
2.29/bittorrent/bittorrent_default.tcl

BitTorrentApp set leave_option -1

# number of peers
set N_P $no_of_peers

# number of seeds
set N_S $no_of_seeds

# factor that download capacity is higher than upload capacity
set C_down_fac 8

# queue size at access links (default 50)
set Q_access 25

# delay
set DelayMin 1
set DelayMax 50
```

```
# file size
set S_F_MB 10

set S_F [expr $S_F_MB * 1024.0 *1024]
set S_C [expr 256.0 *1024]
set N_C [format %.0f [expr ceil($S_F / $S_C)]]

# set the seed for the RNG (0: non-deterministic, 1 - MAXINT (2147483647))
set rng_seed $s

# End of SimulationParameters
set peerCount 0
set FinishedPeers 0

# Create Directory For Data
set p2ptrace    /home/charanpreet/Dropbox/pub_html/Data/bittorrent_data_
append p2ptrace no_peers_
append p2ptrace $no_of_peers
append p2ptrace _no_seeds_
append p2ptrace $no_of_seeds
append p2ptrace _file_size_
append p2ptrace $S_F_MB
append p2ptrace _priority_
if {$priority == 1} {
    append p2ptrace _enabled_
    append p2ptrace $time_delay
    append p2ptrace _
} else {
    append p2ptrace _disabled_
}
append p2ptrace [clock seconds]

exec mkdir $p2ptrace
puts $p2ptrace

#Create NAM object
set nf [open $p2ptrace/bt.nam w]
#Create Tracefile
set f [open $p2ptrace/bt.tr w]

set p2ptrace2 $p2ptrace

exec cp /home/charanpreet/Dropbox/pub_html/Data/bittorrent.tcl $p2ptrace
exec cp /home/charanpreet/Dropbox/pub_html/Data/data.m $p2ptrace
append p2ptrace /log

#Put all data into NAM
$ns namtrace-all $nf

#Put all data in Trace-File
$ns trace-all $f
# set MSS for all FullTCP connections
Agent/TCP/FullTcp set segsize_ 1460
Queue set limit_ $Q_access
```

```
# Seed the default RNG
global defaultRNG
$defaultRNG seed $rng_seed

# Create Connections
proc fully_meshed2 {no_of_peers} {
    global ns peer router C_up C_down_fac DelayMin DelayMax

    set e2eDelayRng [new RNG]
    set e2eDelay [expr round([$e2eDelayRng uniform $DelayMin $DelayMax])]

    # upstream
    $ns simplex-link $peer($no_of_peers) $router $C_up($no_of_peers) [expr
$e2eDelay]ms DropTail
    # downstream
    $ns simplex-link $router $peer($no_of_peers) [expr 1000*1024] [expr
$e2eDelay]ms DropTail

    # do the routing manually between peer and router
    [$peer($no_of_peers) get-module "Manual"] add-route-to-adj-node -default
[$router id]
    [$router get-module "Manual"] add-route-to-adj-node -default
[$peer($no_of_peers) id]

    [$router get-module "Manual"] add-route [$peer($no_of_peers) id] [[$ns
link $router $peer($no_of_peers)] head]

    [$peer($no_of_peers) get-module "Manual"] add-route [$router id] [[$ns
link  $peer($no_of_peers) $router] head]

    return 0
}

proc done {} {
    global app FinishedPeers N_P fh ns nf f p2ptrace2

    incr FinishedPeers

    if {$FinishedPeers == $N_P} {
        for {set i 0} {$i < $N_P} {incr i} {
            $app($i) stop
        }

        close $nf
        close $f

        #Open NAM file
        exec nam $p2ptrace2/bt.nam
        puts [$ns now]
        exit 0
    }
}
```

```
proc priority_tracker {} {
    global  no_of_peers no_of_seeds C_up start_time_ time_delay
    #put the upload rate into an array
    for {set i 0} {$i < $no_of_peers} {incr i} {
        set peer_array($i) $C_up($i)
    }
    #sort the upload rate from lowest to highest
    set j 1
    while {$j == 1}  {
        set j 0
        for {set k 0} {$k < [expr $no_of_peers-1]} {incr k} {
            if {$peer_array([expr $k+1]) < $peer_array($k)} {
                set temp $peer_array($k)
                set peer_array($k) $peer_array([expr $k+1])
                set peer_array([expr $k+1]) $temp
                set j 1
            }
        }
    }
    #finds all the unique values of C_up
    set index 0
    for {set i 0} {$i < [expr $no_of_peers-1]} {incr i} {
        if {$peer_array($i) != $peer_array([expr $i+1])} {
            set unique_array($index) $peer_array($i)
            incr index
        }
    }
    #set the time such that those with the largest value has start time of 0
    set unique_array($index) $peer_array([expr $no_of_peers-1])
    #set start times
    for {set i 0} {$i <= [expr $no_of_peers-1]} {incr i} {
        for {set k 0} {$k <= $index} {incr k} {
            if {$C_up($i) == $unique_array($k)} {
                set start_time_($i) [expr ($index-$k)*$time_delay]
            }
        }
    }
    #start seeds at 0 sec
    for {set i 0} {$i < $no_of_seeds} {incr i} {
        set start_time_($i) 0
    }
}

# create tracker
# Parameters: File Size [B], Chunk Size [B]
set go [new BitTorrentTracker $S_F $S_C]
#$go tracefile $p2ptrace


# uniform start offset for peers
set t_offset_rng [new RNG]
set t_offset [new RandomVariable/Uniform]
$t_offset set min_ 0
$t_offset set max_ [BitTorrentApp set choking_interval]
$t_offset use-rng $t_offset_rng
```

```
set router [$ns node]
$router shape box
$router color blue
$ns at 0.0 "$router label \"The Internet\""

$ns color 1 Red
$ns color 2 Blue
$ns color 3 Green
$ns color 4 Yellow
$ns color 5 Black
$ns color 6 White
$ns color 7 Purple

#call priority function here
if {$priority == 1} {
    priority_tracker
} else {
    for {set i 0} {$i < $no_of_peers} {incr i} {
        set start_time_($i) 0
    }
}

# Create Seeds
for {set i 0} {$i < $N_P} {incr i} {

    # make nodes
    set peer($i) [$ns node]

    # make links
    fully_meshed2 $i

    if {$i < $N_S} {
        set app($peerCount) [new BitTorrentApp 1 $C_up($i) $go $peer($i)]

        $app($peerCount) set super_seeding 0
        $app($peerCount) tracefile $p2ptrace

        # start apps
        $ns at start_time_($i) "$app($peerCount) start"

        $peer($i) color Purple
        $ns at 0.0 "$peer($i) label \"Initial Seed \""

        incr FinishedPeers
    } else {
        set app($peerCount) [new BitTorrentApp 0 $C_up($i) $go $peer($i)]

        $app($peerCount) tracefile $p2ptrace

        # start apps
        $ns at $start_time_($i) "$app($peerCount) start"
        $ns at $start_time_($i) "$peer($i) label \"Peer Entered\""
    }
```

```
    incr peerCount
}

# Run the simulation
$ns at 300.0 "done"
$ns run
```

LOG FILE

The tables below are the log files of the simulation with various numbers of clients in a P2P network. The number of clients varies from 5 peers, 1 seed to 10 peers, 1 seed. In addition, different start times are given to peers. This start times are 1, 5 and 10 seconds. The stop time determines when the entire network is done downloading the single 10mb file.

5 CLIENTS: 1 SEED, 5 PEERS

TABLE 9 - 5 PEERS, 1 SEED - NO PRIORITY

ID	Start_time	first_chunk_time	download_finished	stop_time	download_finished - start time
1	0	0	-1	275.509	-1
2	0	13.7683	274.246	275.509	274.246
3	0	4.35738	275.278	275.509	275.278
4	0	7.37525	275.509	275.509	275.509
5	0	17.6091	271.449	275.509	271.449

TABLE 10 - 5 PEERS, 1 SEED - PRIORITY 1 SECOND

ID	Start_time	first_chunk_time	download_finished	stop_time	download_finished - start time
1	0	0	-1	278.338	-1
2	1	12.7364	278.253	278.338	277.253
3	0	4.36126	273.439	278.338	273.439
4	1	7.37913	277.85	278.338	276.85
5	1	17.6301	278.338	278.338	277.338

TABLE 11 - 5 PEERS, 1 SEED - PRIORITY 5 SECONDS

ID	Start_time	first_chunk_time	download_finished	stop_time	download_finished - start time
1	0	0	-1	272.126	-1
2	5	15.9239	272.126	272.126	267.126
3	0	4.3501	269.831	272.126	269.831
4	5	8.14901	271.864	272.126	266.864
5	5	20.4804	268.546	272.126	263.546

TABLE 12 - 5 PEERS, 1 SEED - PRIORITY 10 SECONDS

ID	Start_time	first_chunk_time	download_finished	stop_time	download_finished - start time
1	0	0	-1	285.027	-1
2	10	17.9019	284.829	285.027	274.829
3	0	4.3501	285.027	285.027	285.027
4	10	12.3914	284.529	285.027	274.529
5	10	22.2047	284.481	285.027	274.481

5 CLIENTS: 3 SEEDS, 5 PEERS

TABLE 13 - 5 PEERS, 3 SEEDS - NO PRIORITY

ID	Start_time	first_chunk_time	download_finished	stop_time	download_finished - start time
1	0	0	-1	96.783	-1
2	0	0	-1	96.783	-1
3	0	0	-1	96.783	-1
4	0	4.19455	88.0406	96.783	88.0406
5	0	9.77674	96.783	96.783	96.783

TABLE 14 - 5 PEERS, 3 SEEDS - PRIORITY 1 SECOND

ID	Start_time	first_chunk_time	download_finished	stop_time	download_finished - start time
1	0	0	-1	94.8938	-1
2	0	0	-1	94.8938	-1
3	0	0	-1	94.8938	-1
4	0	4.01362	89.4785	94.8938	89.4785
5	1	7.68209	94.8938	94.8938	93.8938

TABLE 15 - 5 PEERS, 3 SEEDS - PRIORITY 5 SECONDS

ID	Start_time	first_chunk_time	download_finished	stop_time	download_finished - start time
1	0	0	-1	94.605	-1
2	0	0	-1	94.605	-1
3	0	0	-1	94.605	-1
4	0	4.41018	91.312	94.605	91.312
5	5	9.08417	94.605	94.605	89.605

TABLE 16 - 5 PEERS, 3 SEEDS - PRIORITY 10 SECONDS

ID	Start_time	first_chunk_time	download_finished	stop_time	download_finished - start time
1	0	0	-1	98.8685	-1
2	0	0	-1	98.8685	-1
3	0	0	-1	98.8685	-1
4	0	4.41018	89.1565	98.8685	89.1565
5	10	14.1141	98.8685	98.8685	88.8685

10 CLIENTS: 1 SEEDS, 10 PEERS

Note: This Scenario did not seem to run 100% correctly, but we decided to include it in the appendix anyways since it worked correctly until near the end

TABLE 17 - 10 PEERS, 1 SEED - NO PRIORITY

ID	Start_time	first_chunk_time	download_finished	stop_time	download_finished - start time
1	0	0	-1	322.429	-1
2	0	22.5803	320.507	322.429	320.507
3	0	4.51673	317.51	322.429	317.51
4	0	9.48959	321.555	322.429	321.555
5	0	24.8698	321.547	322.429	321.547
6	0	12.1187	-1	322.429	-1
7	0	16.9306	321.656	322.429	321.656
8	0	29.1623	322.429	322.429	322.429
9	0	27.1566	322.353	322.429	322.353
10	0	26.5542	315.151	322.429	315.151

TABLE 18 - 10 PEERS, 1 SEED - PRIORITY 1 SECOND

ID	Start_time	first_chunk_time	download_finished	stop_time	download_finished - start time
1	0	0	-1	339.7	-1
2	1	27.7536	333.475	339.7	332.475
3	1	7.50729	334.864	339.7	333.864

4	0	4.48941	328.082	339.7	328.082
5	1	27.9666	339.457	339.7	338.457
6	1	12.0077	-1	339.7	-2
7	1	16.828	337.133	339.7	336.133
8	1	22.1986	335.447	339.7	334.447
9	1	30.2819	332.063	339.7	331.063
10	1	25.9845	339.7	339.7	338.7

TABLE 19 - 10 PEERS, 1 SEED - PRIORITY 5 SECONDS

ID	Start_time	first_chunk_time	download_finished	stop_time	download_finished - start time
1	0	0	-1	356.779	-1
2	5	27.5035	355.756	356.779	350.756
3	5	7.6657	346.922	356.779	341.922
4	0	4.4671	355.637	356.779	355.637
5	5	24.5639	356.132	356.779	351.132
6	5	12.1706	355.761	356.779	350.761
7	5	16.9815	356.779	356.779	351.779
8	5	27.5146	346.074	356.779	341.074
9	5	24.0851	-1	356.779	-6
10	5	28.0051	353.436	356.779	348.436

TABLE 20 - 10 PEERS, 1 SEED - PRIORITY 10 SECONDS

ID	Start_time	first_chunk_time	download_finished	stop_time	download_finished - start time
1	0	0	-1	360.209	-1
2	10	27.7274	351.625	360.209	341.625
3	10	12.5616	348.297	360.209	338.297
4	0	4.4671	360.209	360.209	360.209
5	10	34.8367	358.266	360.209	348.266
6	10	17.6835	358.674	360.209	348.674
7	10	22.952	-1	360.209	-11
8	10	27.3632	349.754	360.209	339.754
9	10	42.6275	354.44	360.209	344.44
10	10	29.1973	349.345	360.209	339.345

10 CLIENTS: 5 SEEDS, 10 PEERS

TABLE 21 - 10 PEERS, 5 SEEDS - NO PRIORITY

ID	Start_time	first_chunk_time	download_finished	stop_time	download_finished - start time
1	0	0	-1	118.708	-1
2	0	0	-1	118.708	-1
3	0	0	-1	118.708	-1
4	0	0	-1	118.708	-1
5	0	0	-1	118.708	-1
6	0	5.93705	114.619	118.708	114.619
7	0	15.6919	118.071	118.708	118.071
8	0	26.4805	115.687	118.708	115.687
9	0	23.8726	118.708	118.708	118.708
10	0	25.4215	116.26	118.708	116.26

TABLE 22 - 10 PEERS, 5 SEEDS - PRIORITY 1 SECOND

ID	Start_time	first_chunk_time	download_finished	stop_time	download_finished - start time
1	0	0	-1	112.33	-1
2	0	0	-1	112.33	-1
3	0	0	-1	112.33	-1
4	0	0	-1	112.33	-1
5	0	0	-1	112.33	-1
6	1	13.2152	112.33	112.33	111.33
7	1	7.93149	111.994	112.33	110.994
8	0	3.69505	110.626	112.33	110.626
9	1	24.7416	111.535	112.33	110.535
10	1	17.4676	111.307	112.33	110.307

TABLE 23 - 10 PEERS, 5 SEEDS - 5 SECONDS

ID	Start_time	first_chunk_time	download_finished	stop_time	download_finished - start time
1	0	0	-1	111.765	-1
2	0	0	-1	111.765	-1
3	0	0	-1	111.765	-1
4	0	0	-1	111.765	-1
5	0	0	-1	111.765	-1
6	5	10.8781	108.697	111.765	103.697
7	5	21.4159	110.345	111.765	105.345
8	0	6.88248	105.088	111.765	105.088
9	5	24.8195	111.765	111.765	106.765
10	5	20.5582	109.527	111.765	104.527

TABLE 24 - 10 PEERS, 5 SEEDS - PRIORITY 10 SECONDS

ID	Start_time	first_chunk_time	download_finished	stop_time	download_finished - start time
1	0	0	-1	118.203	-1
2	0	0	-1	118.203	-1
3	0	0	-1	118.203	-1
4	0	0	-1	118.203	-1
5	0	0	-1	118.203	-1
6	10	15.5714	116.189	118.203	106.189
7	10	19.7144	118.203	118.203	108.203
8	0	6.90978	116.741	118.203	116.741
9	10	14.3465	117.786	118.203	107.786
10	10	28.8848	115.676	118.203	105.676