Studying the average download times with respect to different piece selection algorithms in a BitTorrent-like P2P Network

- Agarwal, Kshitij

- Louzolo-Kimbembe, Igor

Overview of the topic area

BitTorrent - It is a protocol that supports P2P file sharing. It is used to distribute a large amount of data over the internet.

Basic Terms in a nutshell:-

- **BitTorrent Portal** - A central server where certain users can publish their .torrent files and other users can download them. A typical example of a portal can be a website.
- **BitTorrent Swarm** - A set of peers that all download a particular content.
- **BitTorrent Tracker** - The server that keeps track of all the clients that form a swarm for a file.
- **BitTorrent Client**
    - Seeder - Has the complete file.
    - Leecher - Doesn't have the complete file, but instead it may have some pieces of the file.
- A **.torrent file** contains:-
    - Content name
    - File Size
    - Chunks
        - Number of chunks
        - Size of chunks
    - Torrent infohash - Unique identifier of the swarm associated with a .torrent file.
    - IP address of the tracker managing the swarm.
- **Pieces**
    - Made of blocks - Blocks might also be divided into sub-blocks.
    - Two or more blocks make up a whole piece.

How does a peer joins and leaves the swarm:-

- User gets a .torrent file and runs it in a BitTorrent client software
    - The software picks one or more tracker mentioned in the .torrent file
- The tracker is contacted using the IP address as specified in the .torrent file.
    - Peer uses a request in the BitTorrent protocol named 'Announce started'
- Response received

    - The response of the announce started request contains the peer neighborhood.
    - Peer neighborhood contains:-
        - Some random seeds and peers(typically between 40 - 200)
        - The IP address of all the seeds and peers returned are also provided so that the receiving peer can contact them.
- Leaving the swarm
    - Peer uses a request in the BitTorrent protocol named 'Announce stopped'

Few important points of how a file gets exchanged:-

- **Peer Wire Protocol**
    - The two communicating peers do a handshake between them.
    - Understanding bitfields:-
        - Current download information of the file concerned
        - Specifically the pieces of the file, that is, which piece is complete and which is not
    - BITFIELD message
        - Exchanged before having any kind of file transfer
        - Content exchanged are bitfields
    - Every peer knows about the file download information of all its neighbors.

    - HAVE Message
        - When a new piece is completely downloaded by a peer, the peer has to notify everybody in the peer neighborhood about the same.
        - This notification does not involve the whole bitfields exchange again as only the change in bitfields has to be notified.

Hypothesis

Weighty piece selection algorithm reduces the average download time of a particular file with respect to the rarest first piece selection algorithm.

Average Download Time:-

The average download time in our simulation is defined as follows:-

- The average download time of all the peers in a particular trial.

- The average download time for individual peer averaged over the number of trials.

Understanding the Algorithms

- Rarest First Piece Algorithm
    - It favors the download of pieces that are rare in the current peer neighborhood.
    - Purpose:-
        - Overcomes the last piece problem by increasing the piece availability of rare pieces.
    - Rarest First Piece calculation
        - Every peer maintains the count of copies of all the pieces available in the peer neighborhood

        - The peer having a missing piece, also knows that how low is the availability of the piece in the network
        - Peer chooses a particular missing piece for download - that is available with some other peer(s), but its total availability is relatively very low with respect to other missing pieces.

- Weighty Piece Selection Algorithms
    - Purpose
        - It is designed for the purpose of reducing the average download time.
    - Weight assignment of each missing piece
        - It takes into consideration - all the peers that miss the same piece that a particular peer wants to request.
        - Sum of downloaded pieces of all the peers, selected above, is calculated individually.
        - Sum of individual sums, calculated above, is calculated.
        - The added sum is the weight of the missing piece.

Approach taken to prove/disprove the hypothesis

- Simulation of BitTorrent-like peer to peer protocol.
    - First the simulation runs using the rarest first piece selection algorithm.
    - Subsequently the simulation runs using the weighty piece selection algorithm.
    - All other things are kept the same except the piece selection algorithm.

Measurements

- Time starts at the beginning of the simulation.
- Time is measured again when every client has the all the pieces of the file.

Analysis of Research Papers

First Research Paper: Improving the download time of BitTorrent –like systems:

Advantages of the weighty piece selection algorithm over the rarest first piece selection algorithm.

The central issue addressed in this research paper is that of the performance of BitTorrent-like systems, in particular the average download time of all the peers in a P2P network.
We’ve previously seen that BitTorrent is a file sharing protocol that makes use of the local rarest first content distribution technique to enable the download of large amounts of data in a scalable manner over a P2P network. However, there is a lack of central coordination and scheduling in a BitTorrent-like network, that is, while increasing the network performance by enabling concurrent uploads and downloads of large volumes of data, BitTorrent doesn’t have control over its clients in terms of the overall download time and the total elapsed time. This can be beneficial especially when the number of peers in the network increases (handling a large

number of clients in a network would be a difficult task), but it also has a performance issue as these metrics (the download time and the total elapsed time) are not considered, meaning that narrow-bandwidth peers can suffer a long delay while downloading a particular file.

Like the rarest first piece algorithm, the weighty piece selection algorithm is based on the idea that a peer should start by downloading missing pieces from its neighbors that also request it. However, in the weighty piece selection algorithm a weight is assigned to a missing piece according to the sum of all neighbors' downloaded pieces, which means that the highest weight missing piece is obtained from the requesting neighbors that have plenty of other pieces. This way, a plentiful neighbor can download a missing piece from its peer that has few pieces, reducing the download time.

The formulae below describe the computation of the rarest first piece and a weighty piece.

Notations	Definitions
Mi	The set of pieces that the peer is missing
Di	The set of pieces that peer i has downloaded
N(i)	The set of neighbors of i
L(Di)	The number of pieces downloaded
B	The set of all the pieces
α	The parameter of the algorithm

The computation of the rarest piece is described by the following formula.

$$\forall\, m \in (Dn \cap Mr)$$

$$F(m) = \{ \sum_{\substack{i \in N(r) \\ 0}} L(Di)^{\propto = 0}\ if\ m \in Mi$$

Here, we set the parameter α = 0 meaning that the rarest piece is determined by the number of peers that request it.

The computation of the weighty piece algorithm is described by the almost-identical formula:

$$\forall m \in (Dn \cap Mr)$$

$$F(m) = \{ \sum_{\substack{i \in N(r) \\ 0}} L(Di)^{\propto=1} \; if \; m \in Mi$$

Here, we set the parameter α = 1, meaning that the weight of a piece is assigned according to the number of requesting neighbors 'downloaded pieces.

At this point we cannot clearly say that the weighty piece algorithm is more efficient that the rarest first piece algorithm, we need to verify this by using the above formulae to implement a peer that selects its pieces according to the number of its peer that request the same piece, and another peer that selects its pieces according to the total number of pieces downloaded by its peers that also request the same piece.

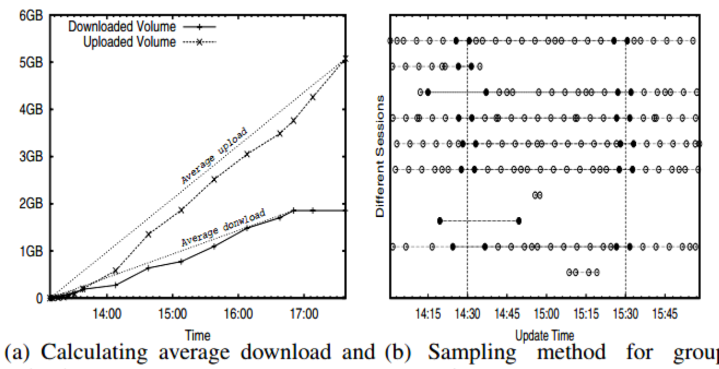
Second Research Paper: Understanding Peer-level Performance in BitTorrent: A Measurement Study

In this paper we talk about the distribution of observed performance of a peer among participating peers in a torrent.
This article provides a new approach for measuring peer-level performance in a BitTorrent-like system with a representative characterization of the performance. This allows to unravel the peer-level and group-level properties that are related to peer-level performance in a BitTorrent-like system.

The peer-level properties considered in this article are the upload and download rates of individual peers in a torrent.

- A Tracker log file is used to derive peer-level properties.
- The log file contains session events associated with each peer in the torrent.
- The average download and upload rates between two updates is measured
- The average download and upload rates for all the session is measured.
- The peer level performance is the ratio of the average download rate to the maximum download rate. Since the maximum download rate cannot be obtained from the tracker log, we need to get the maximum value of the download rate per interval and ratio it to the average download value to get the peer level performance.

- The peer level performance is highly correlated to the average upload rate (contributions). This means that the Tit-For-Tat mechanism is the primary factor that affects performance in a BitTorrent –like system.



(a) Calculating average download and upload rate (b) Sampling method for group properties

Fig. 1. Capturing peer- and group-level properties

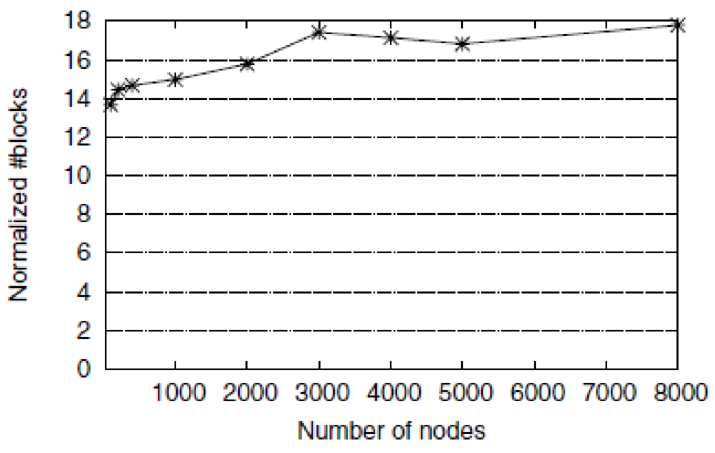
We now know from second research paper that the performance of a BitTorrent-like system partly depends on the average download rate, which is a function of time. This means that if we prove our hypothesis that the weighty piece selection algorithm reduces download time compared to the local rarest first strategy, then not only will we have minimized the download time, but we will also have increased the performance of the network!

Third Research Paper: Analyzing and Improving a BitTorrent Network's performance Mechanisms:

In order to understand BitTorrent performance, a simulation-based approached is adopted.
Description of the simulator:

The simulator models peer activities (joins, leaves, block exchange) and BitTorrent's mechanisms (Tit-For-Tat, local rarest first…).

The simulator associates a downlink and an uplink bandwidth for each node.
As the size of the system increases, BitTorrence's performance scales very well both in terms of the bandwidth utilization and the work of the seeds.

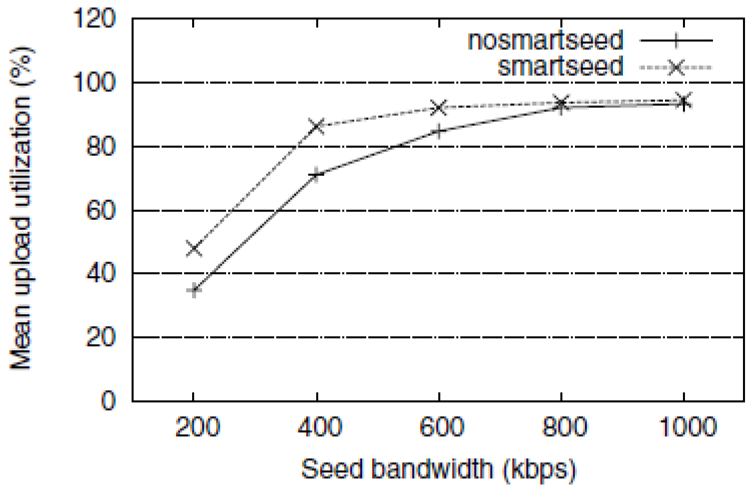


(b)

Number of seeds and bandwidth of seeds.

In this part the performance of BitTorrent is measured with respect to the number of seeds and their bandwidth.

The upload utilization of a "nosmartseed" tends to stay below optimal as new peers are added to the network, no matter how much the its bandwidth is increased. This is due to the fact that blocks are served prematurely (some peers request the same blocks asynchronously).
Applying a "smartseed" policy helps fix this problem. A "smartseed" does not choke a leecher until it has received the complete. A smartseed serves the blocks it at served the least among the ones the leecher is looking for. This improves the upload utilization of the seed.



(c)

This implies that, although changing the local rarest first policy, the “smartseed” policy increases the system’s performance in terms of upload utilization.

Block choosing policy and Node degree

In this analysis, it is assumed that the “smartseed” policy is used.
The general trend is that the upload utilization increases with both the seed bandwidth and the node degree.

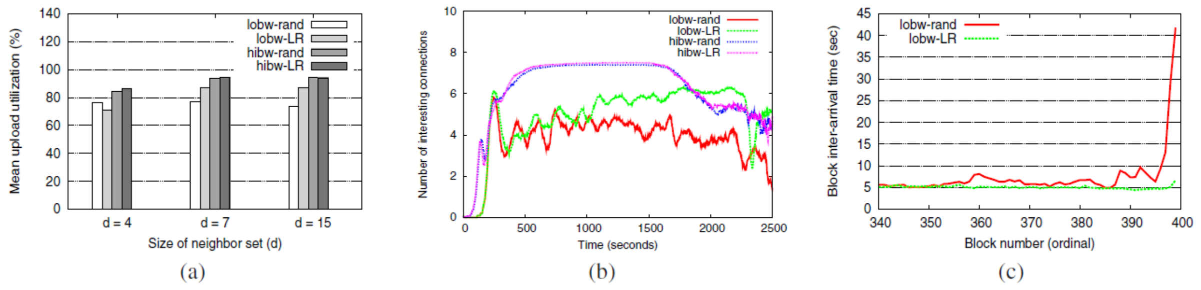


Fig. 3: (a) Upload utilization for LRF and Random policies for different values of the node degree, d. (b) Variation of the number of *interesting* connections over time for $d = 7$. (c) Inter-arrival times for blocks at the tail end of the file; each point represents the mean time to receive the k^{th} block, where the mean is taken over all nodes.

The results of this research paper are the following (as stated in the article):

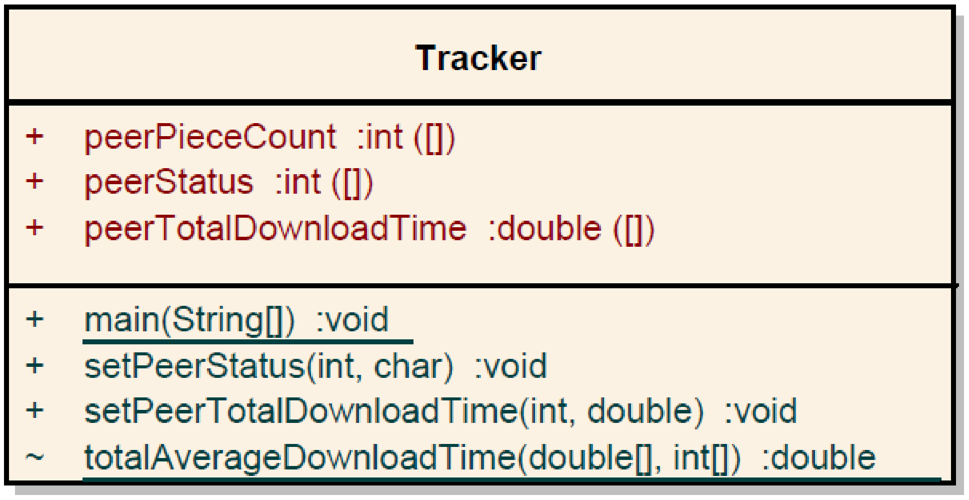
BitTorrent’s rate-based Tit-For-Tat (TFT) policy fails to prevent unfairness across nodes in terms of volume of content served.
The combination of pairwise block-level TFT (Section VIIB) and the bandwidth matching tracker (Section VII-C) almost eliminates the unfairness of BitTorrent with a modest decrease in utilization.
It is critical to conserve seed bandwidth, especially when it is scarce; it is important that the seed node serve unique blocks at first (which it alone can do) to ensure diversity in the network, rather than serve duplicate blocks (a function that can be performed equally well by the leechers).
The Local Rarest First (LRF) policy is critical in eliminating the “last block” problem and ensuring that new leechers quickly have something to offer to other nodes.

How we plan to use third research paper’s results in our investigation:
The last result of this research paper is interesting; it states that the local rarest first policy eliminates the “last block” problem and ensures that new leechers quickly have something to share to other nodes, which increases the system’s performance.

However, our goal is to replace the LRF with the WLRF strategy. In doing so, we'll verify whether that affects the system's performance, that is, whether the system behaves equally well or is more effective.

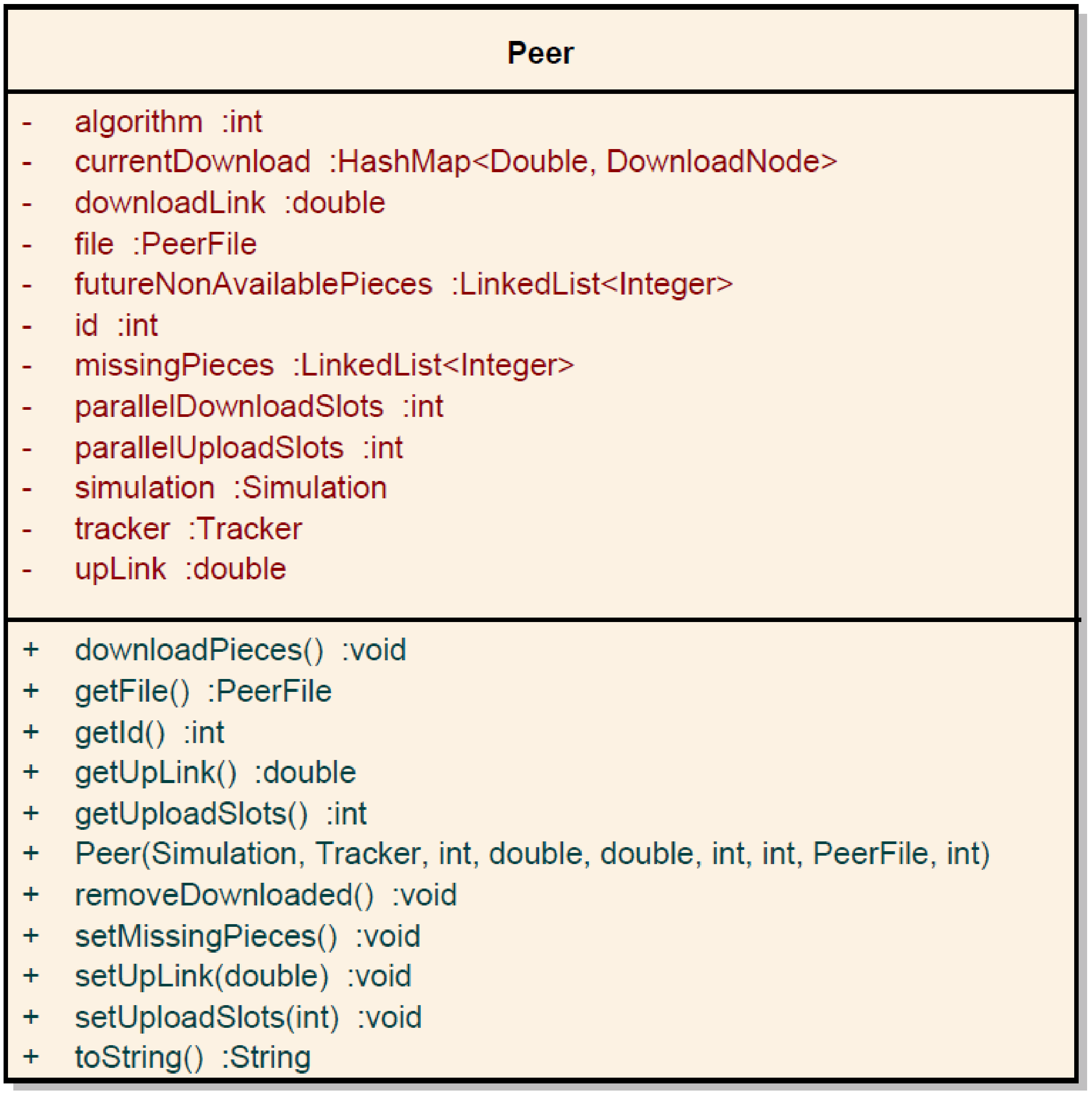
The design of the software

- Tracker Design



Tracker
+ peerPieceCount :int ([])
+ peerStatus :int ([])
+ peerTotalDownloadTime :double ([])
+ main(String[]) :void
+ setPeerStatus(int, char) :void
+ setPeerTotalDownloadTime(int, double) :void
~ totalAverageDownloadTime(double[], int[]) :double

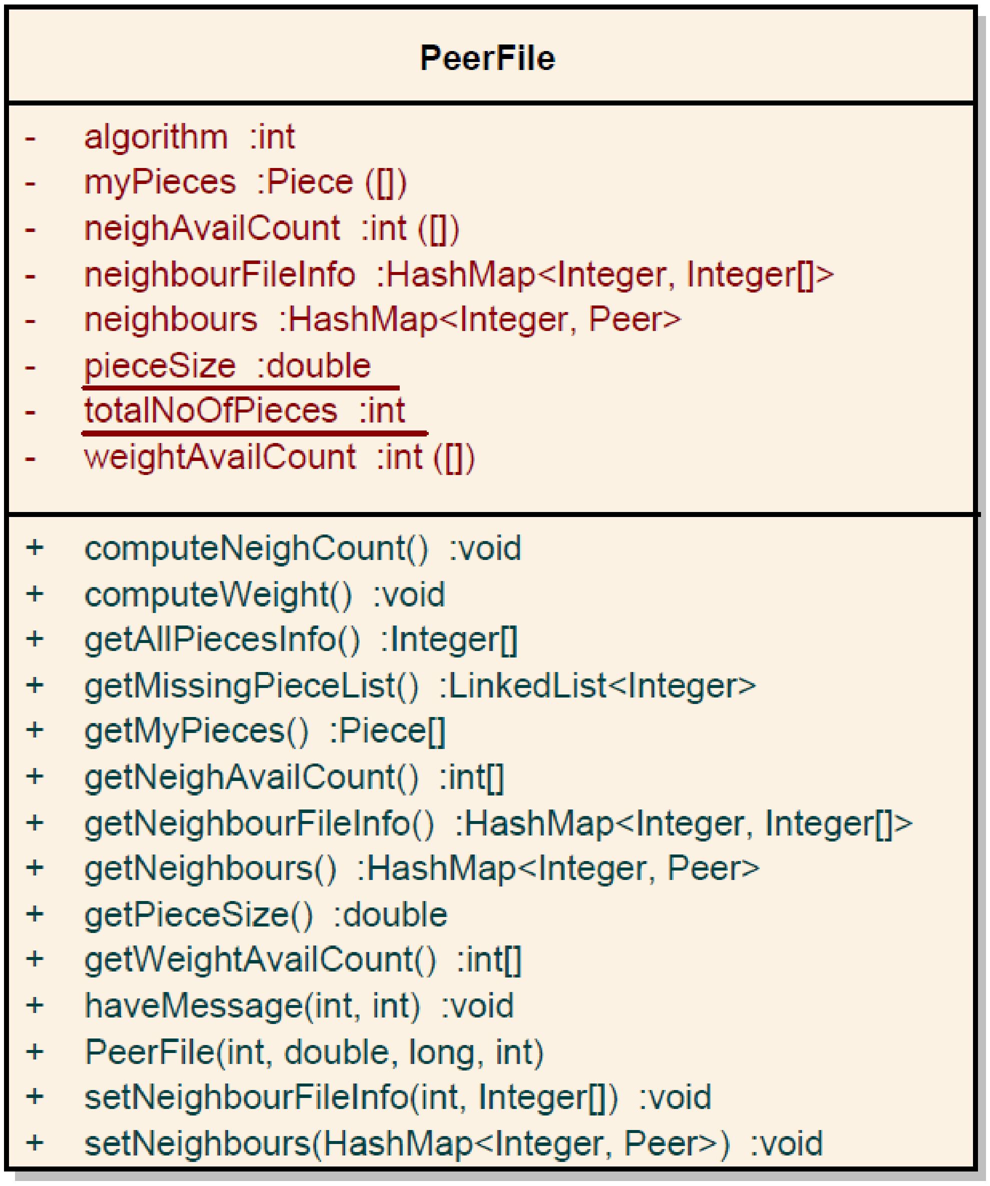
- Peer Design



```
Peer
- algorithm :int
- currentDownload :HashMap<Double, DownloadNode>
- downloadLink :double
- file :PeerFile
- futureNonAvailablePieces :LinkedList<Integer>
- id :int
- missingPieces :LinkedList<Integer>
- parallelDownloadSlots :int
- parallelUploadSlots :int
- simulation :Simulation
- tracker :Tracker
- upLink :double

+ downloadPieces() :void
+ getFile() :PeerFile
+ getId() :int
+ getUpLink() :double
+ getUploadSlots() :int
+ Peer(Simulation, Tracker, int, double, double, int, int, PeerFile, int)
+ removeDownloaded() :void
+ setMissingPieces() :void
+ setUpLink(double) :void
+ setUploadSlots(int) :void
+ toString() :String
```

- Peer File

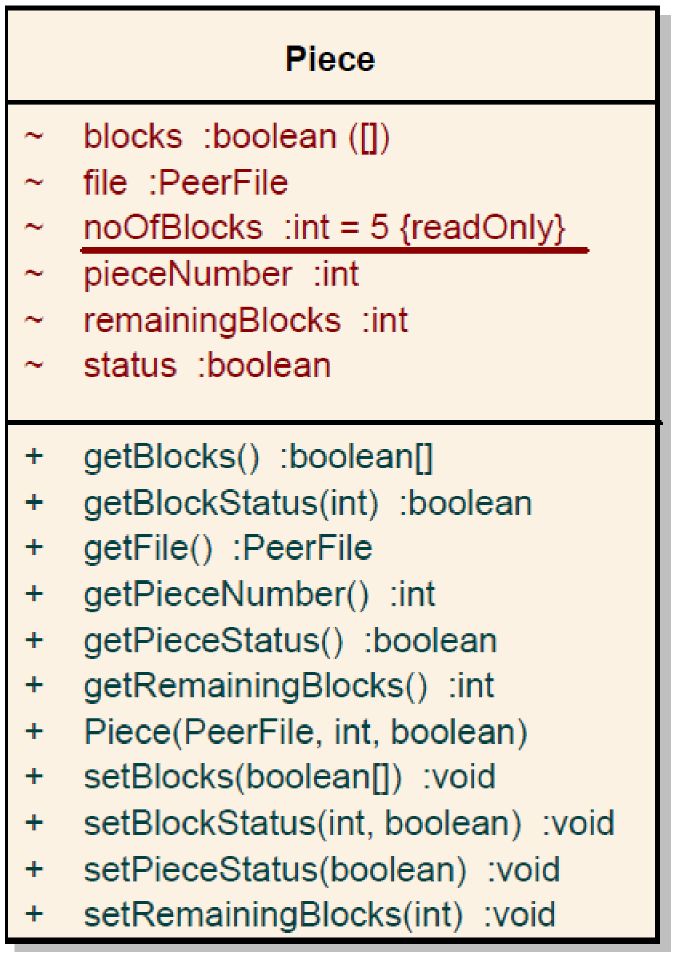


PeerFile

```
-    algorithm  :int
-    myPieces  :Piece ([])
-    neighAvailCount  :int ([])
-    neighbourFileInfo  :HashMap<Integer, Integer[]>
-    neighbours  :HashMap<Integer, Peer>
-    pieceSize  :double
-    totalNoOfPieces  :int
-    weightAvailCount  :int ([])

+    computeNeighCount()  :void
+    computeWeight()  :void
+    getAllPiecesInfo()  :Integer[]
+    getMissingPieceList()  :LinkedList<Integer>
+    getMyPieces()  :Piece[]
+    getNeighAvailCount()  :int[]
+    getNeighbourFileInfo()  :HashMap<Integer, Integer[]>
+    getNeighbours()  :HashMap<Integer, Peer>
+    getPieceSize()  :double
+    getWeightAvailCount()  :int[]
+    haveMessage(int, int)  :void
+    PeerFile(int, double, long, int)
+    setNeighbourFileInfo(int, Integer[])  :void
+    setNeighbours(HashMap<Integer, Peer>)  :void
```

- Piece


Piece
~ blocks :boolean ([])
~ file :PeerFile
~ noOfBlocks :int = 5 {readOnly}
~ pieceNumber :int
~ remainingBlocks :int
~ status :boolean
+ getBlocks() :boolean[]
+ getBlockStatus(int) :boolean
+ getFile() :PeerFile
+ getPieceNumber() :int
+ getPieceStatus() :boolean
+ getRemainingBlocks() :int
+ Piece(PeerFile, int, boolean)
+ setBlocks(boolean[]) :void
+ setBlockStatus(int, boolean) :void
+ setPieceStatus(boolean) :void
+ setRemainingBlocks(int) :void

- Current Software Design
  - Main Program - Tracker
    - Holds all the peer objects.

  - Simulation of the seeds and leechers
    - The peer objects hold the file object for maintaining the state of the file being downloaded.
    - We simulate only one file to study the download rates.
    - The peer object hold the information of peer neighborhood.

  - Simulation of the bandwidth

    - Download and upload rates of each peer are assigned.

- Use of RIT's Parallel Java Library
    - Usage of package "edu.rit.sim" for simulating the download of a piece of a file.
    - Achieving the simulation of downloading a piece by the use of simulated time.
        - Future events are set for denoting the completion of the download of a particular piece.
        - During the download, the available bandwidth and number of slots are being affected.
        - No actual file piece transfer is happening.

- Implementation of achieving the effect of Tit-for-Tat Policy
    - The tit-for-tat policy aims at matching the peers according to similar bandwidths.
    - Similar bandwidth means the download speed of the peer requesting a piece and the upload speed of the peer that provides the piece should be nearly the same.

- Implementation of strategic peer
    - Strategic peer means that the peer leaves the system as soon as the peer completes downloading its missing pieces.
    - In the current software, as soon as the peer has no missing piece, its status is set as seed. Then, it stops contributing anything to the simulation and no calculations are hampered.
        - For Rarest Piece First Algorithm - Relatively, it doesn't matter if every piece count is one extra for calculation.
        - For Weighty Piece First Algorithm - If no pieces are missing within a peer's file, the weight calculation is independent of this peer.

- Assumptions
    - No Last piece problem.
        - The reason for the above is that we have provided with one seed in every peer neighborhood.
    - A peer downloads one piece at a time.
    - Control messages bandwidth is separate and thus does not interfere with the download and upload bandwidth of the pieces.
    - Download time calculation starts from a snapshot of the system
        - Understanding a snapshot of a system:-
            - All the peers and one seed is available before the download measurements start.

    - This will be a random situation in a real environment where some peers would be running and at any instant the calculation can begin.
- No more peers are added or removed once the simulation starts running.
- Peer have some random pieces of the file initially(same pieces for both the versions)

<u>Details of implementation:-</u>

<u>Tracker.java</u>
The tracker program has some functionality of a tracker. It also contains a main function that helps the simulation of a BitTorrent-like peer to peer network.

Tracker program first runs the simulation for Rarest First Piece Strategy and collects the results. Then, the program runs the simulation for Weighty Piece First Strategy and collects the results.

<u>File Creation:-</u>

The file object is created using PeerFile.java, whose constructor has the arguments containing seed for random number generator and the algorithm for computations of piece availability or weights calculations.

We have generated the file for the initial seed by providing it the argument of -1 for the seed.
Argument of -1 in the seed argument of the constructor of PeerFile generates the file object with all the pieces as complete.
Rest of the file objects for the remaining peers are provided with a random seed while creation of the file so that each file object within different peer has different missing piece. The important thing to note is that, corresponding to two different algorithms, the missing pieces in a file should be the same. That has been taken care of.

<u>Peer Creation:-</u>
The peer object is created using Peer.java, whose constructor has the arguments as
- the reference to the file object that was created before creating the peer object and the algorithm that this peer will use to download the pieces.

We generate a peer object with ID ‘0’ that will hold the reference to the seed file.
Counting this as one of the peers, we generate the number of peers provided by the user via command line arguments subtracted by 1.

There are two kinds of peer objects that have to be created

- Seed
- Leecher

Seed in every peer neighborhood:-

According to our major assumption that there will be no last piece problem while simulating this BitTorrent-like peer to peer network - every peer neighborhood has been provided with a seed. It is similar to a scenario where a software provider wants his clients to get the update of his software and wants that there should never be a problem like a client falling short of a particular piece in obtaining the update just because that piece is not available anywhere in the network.

Peer Upload and Download Rates:-

While generating peers, we consider scenario of peers having different download and upload rates. So we can have variety of bandwidths.
For this purpose whenever the peers are generated in each trial we assign them pseudo random rates and this is same for each corresponding trial when the other algorithm runs.
That is, bandwidth remains same for both the algorithms; given a particular peer id in both the cases with the same trial number, the upload and download speeds are the same.

Neighbors for each of the peers:-

Firstly, the peer neighborhood is with respect to a particular file and not with respect to a particular peer on the internet. Secondly, we set the random neighbors to peer file objects, keeping in mind that initial seed is available in every peer neighborhood. Before doing this we also create the neighborhood of initial seed - every other peer is in its neighborhood.

After the neighborhoods are created, the bitfields are exchanged among the peers.

Before actually running the simulation, we need that the neighbors compute the availability of all the pieces in the peer neighborhood in the case of rarest first piece selection strategy and the weights should be computed in the case of weighty piece first selection strategy.

We set the initial missing pieces for the peer just for a start and then we run the simulation so that the download of the pieces is simulated. The results are record and displayed.

Note: The **Piece.java** specifies the methodology for downloading a piece in a way that it is downloaded block by block. But this is just an extension for future work. Currently the pieces are downloaded in its entirety, i.e. all blocks at once.

Peer.java

When the simulation runs, the function that is first executed is **downloadPieces()**. It looks into the missing pieces, then gets a list of providers that can provide it with the missing piece according to some parameters and then when it selects the peer from which it would download that piece, it puts an entry to the current download HashMap that represents a piece being downloaded, with its key as finish time of the download of that piece. The simulation is instructed(method's name - **removeDownloaded()**) to take this piece out of the HashMap at exactly its finish time and neighbors are notified of the same, i.e. the metadata in the peer file objects are updated.

This marks the finish of download of a particular piece. After the download of a particular piece is finished, a new missing piece starts getting downloaded.
We also have taken care of the scenario that due to some limitation of bandwidth and slots, a particular piece might not be downloaded as scheduled by the algorithm. At such times, the piece is pushed into a future list that would be downloaded after all the missing pieces that can be downloaded in order given by the algorithm. When both these lists become empty, the download ends.

Developer's Manual

Download and extract:-

RIT's Parallel Java Library
http://www.cs.rit.edu/~ark/pj.shtml
RIT's Computer Science Course Library
http://www.cs.rit.edu/~ark/cscl.shtml

- Extract the zip file containing all the programs to a directory of your choice

- Set the class path first to the current directory then to the parallel java library and then to the computer science course library.

- Compile all the programs then by
javac *.java

User's manual

Usage: java Tracker <No Of Peers> <Neighborhood Size> <No of chunks> <Piece Size> <Seed Upload Rate> <Peer Download Rate> <Peer Upload Rate> <Peer Download Slots> <Peer Upload Slots> <Number of Trials> <Master Seed>

The command line arguments are explained as follows:-

<No Of Peers>	The number of peers in the simulation
<Neighborhood Size>	Number of peers in the peer neighborhood
<No of chunks>	The number of chunks(file pieces) in the file

<Piece Size>	The size of each chunk(file piece)
<Seed Upload Rate>	The upload rate of the seed
<Peer Download Rate>	The maximum peer download rate that the peer can have
<Peer Upload Rate>	The maximum peer upload rate that the peer can have
<Peer Download Slots>	The maximum number of peer download slots
<Peer Upload Slots>	The maximum number of peer upload slots
<Number of Trials>	The number of trials for collecting the results
<Master Seed>	The initial seed that helps in generating pseudo random numbers

Measurement of the data collected

1) Rarest First Piece Selection Algorithm:
   a) Total average download time for all the peers in a trial

Trial Number	Total Average Download Time
1	5.11
2	4.93
3	5.26
4	5.76
5	6.29
6	5.44
7	7.31
8	5.15
9	6.76
10	4.76
11	6.44
12	6.24
13	5.06
14	5.97
15	6.64
16	5.66
17	7.07
18	6.94
19	5.28
20	4.44
21	6.85
22	5.80
23	5.14
24	6.16
25	6.39
26	5.16
27	5.80
28	4.87
29	6.56
30	5.32

b) Data for the average download time of individual peer divided by the number of trials:

Peer Number	Average Download Time
0	-
1	0.20
2	0.15
3	0.19
4	0.18
5	0.18
6	0.18
7	0.25
8	0.22
9	0.14
10	0.21
11	0.18
12	0.17
13	0.19
14	0.19
15	0.20
16	0.17
17	0.18
18	0.26
19	0.18
20	0.19
21	0.23
22	0.26
23	0.19
24	0.22
25	0.19
26	0.21
27	0.20
28	0.19
29	0.18

2) Weighty Piece Selection Algorithm:

   a. Total average download time for all the peers in a trial.

Trial Number	Total Average Download Time
1	5.25
2	4.69
3	5.36
4	5.25
5	6.21
6	5.27
7	7.54
8	5.01
9	7.09
10	5.21
11	6.15
12	6.18
13	4.98
14	5.54
15	6.75
16	5.60
17	7.43
18	6.69
19	5.12
20	4.23
21	6.91
22	5.63
23	5.06
24	6.43
25	6.05
26	5.43
27	6.06
28	4.57
29	6.35
30	5.35

b. Data for the average download time of individual peer divided by the number of trials:

Peer Number	Average Download Time
0	-
1	0.22
2	0.15
3	0.19
4	0.18
5	0.18
6	0.17
7	0.24
8	0.21
9	0.15
10	0.22
11	0.17
12	0.17
13	0.18
14	0.19
15	0.19
16	0.16
17	0.19
18	0.25
19	0.16
20	0.20
21	0.21
22	0.27
23	0.18
24	0.24
25	0.20
26	0.19
27	0.19
28	0.20
29	0.19

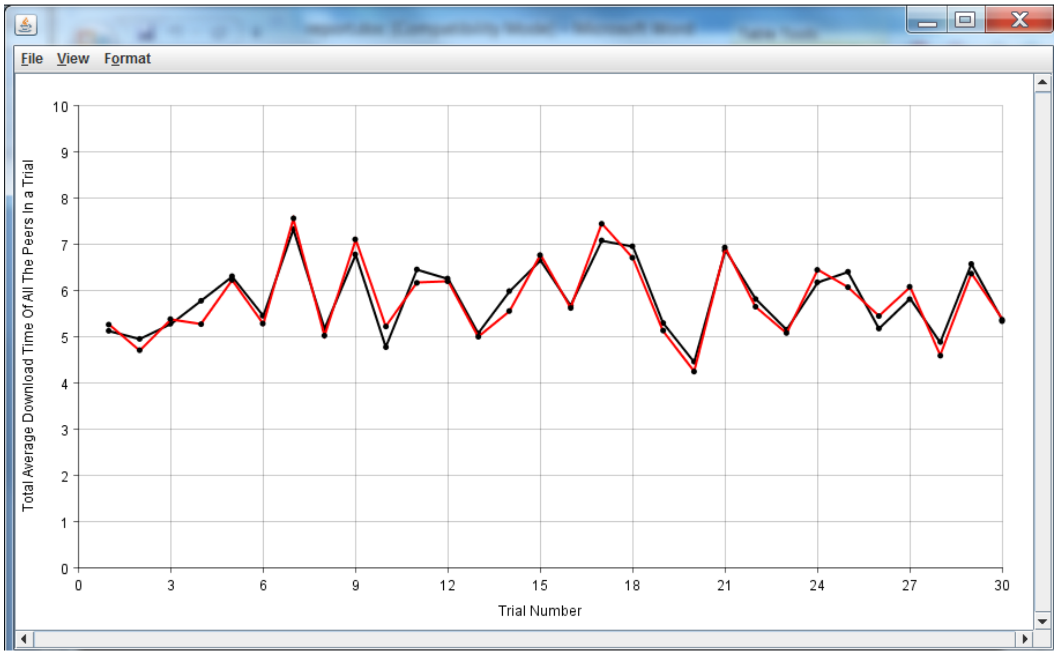
Data Analysis

Parameters used in this simulation:

Number of peers	30
Neighborhood size	10
Number of chunks	100
Piece size	10
Seed Upload rate	300
Peer download rate	250
Peer upload rate	200
Peer download slots	10
Peer upload slots	10
Number of trials	30
Master seed	123456

Analysis of the total average download time of all the peers in a trial.

The data collected reveal that regardless of the number of nodes, the number of trials and that of the neighbors, the individual average download time in both cases (rarest first piece and weighty piece selection algorithm) follow a same, quasi-periodic pattern; the average time values alternately go up and down. This can be explained not by the tit-for-tat policy, but by one of its effects. Initially the peer downloads the missing piece its algorithm has determined with the highest priority (rarest piece or weightiest piece), taking some time. Once the peer has downloaded the missing piece, it starts sharing (uploading) it to its neighbors that are missing the piece. Since the tit-for-tat policy favors peers that share their pieces, the next time the peer downloads a piece, it will select the neighbor that has a high upload rate, allowing it to download the piece in less time. This repeats as the peer selects the next missing piece (rarest or weightiest).

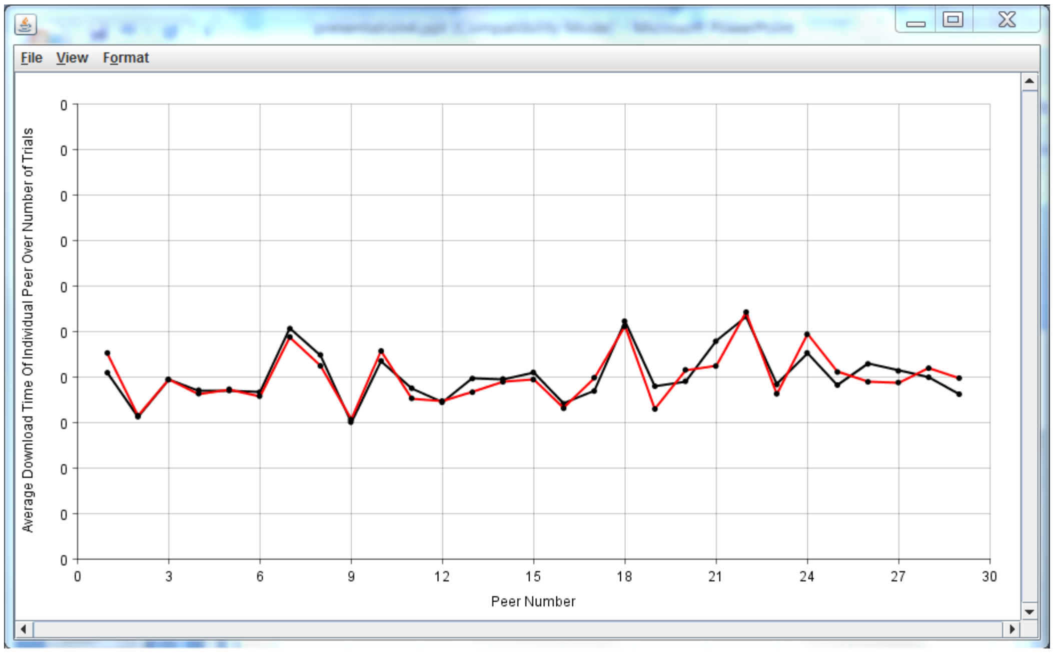


We can observe that the tit-for-tat policy increases the upload capacity of the network in a scalable manner. We can also observe that when the tit-for-tat rule is enforced, the piece selection algorithm matters less.

Analysis of the average download time of individual peer divided by the number of trials

In this case we observe an almost similar pattern; the average time values alternately go up and down. This implies two things. First, the tit-for-tat policy applies in a scalable manner (for any number of peers). Then, the average download time is not a function of the piece selection strategy. Considering the strategy of the weighty piece selection algorithm, it could possibly reduce the average download time of all the peers. However, that would require to not implement the tit-for-tat policy, which would then decrease the upload capacity of the overall network.

In summary, we can say that the result obtained from the simulation disprove the hypothesis that the weighty piece selection algorithm reduces the average download time with respect to the rarest first piece selection algorithm.



A discussion of possible future work

- The possible future work includes the following:-

- Currently one piece is being downloaded at a time. It has to be extended to multiple pieces at a time. So the download slots don't matter for the current version of the program.
- Full piece download to block by block download. Provision for storing metadata for piece download block by block has been provided so that extending the project is easy.
- Control messages bandwidth is assumed to be separate and thus not interfering with the download and upload bandwidth of the file transfer. That bandwidth can also be merged and simulated.

Learning from the project

- This team research investigation has allowed us to gain a deep insight of BitTorrent-like networks.
- Understanding the concepts like why BitTorrent is used to distribute large scale data over internet.
- In addition, we've gained experience on how to read and analyze scientific articles, draw hypothesis and verify them using simulation programs.

A statement of what each individual team member did on the project

- Agarwal, Kshitij
    - Software design
    - Software Coding

- Louzolo-Kimbembe, Igor
    - Analysis of all the Research Papers 1, 2 and 3.
    - Analysis of the data collection from the simulation.

List of references

- Chi-Jen Wu; Cheng-Ying Li; Jan-Ming Ho; , "Improving the Download Time of BitTorrent- Like Systems," Communications, 2007. ICC '07. IEEE International Conference on 24-28 June 2007,page 1125-1129, doi: 10.1109/ICC.2007.191
    - http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=4288862

- Amir H. Rasti; Reza Rejaie; , "Understanding Peer-level Performance in BitTorrent: A Measurement Study," Computer Communications and Networks, 2007. ICCCN 2007. Proceedings of 16th International Conference on 13-16 Aug. 2007,page 109 – 114
    - http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=4317805

- Bharambe, A. R.;Herley, C.; Padmanabhan, V. N.; , "Analyzing and Improving a BitTorrent Networks Performance Mechanisms," INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings, April 2006, page 1-12
    - http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=4146981

- RIT's Parallel Java Library – Developed by Prof. Alan Kaminsky
    - http://www.cs.rit.edu/~ark/pj.shtml

- RIT's Computer Science Course Library – Developed by Prof. Alan Kaminsky
    - http://www.cs.rit.edu/~ark/cscl.shtml