ENHANCING BITTORRENT-LIKE PEER-TO-PEER CONTENT DISTRIBUTION WITH CLOUD COMPUTING

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Zhiyuan Peng

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

Haiyang Wang

November 2018

Abstract

BitTorrent is the most popular P2P file sharing and distribution application. However, the classic BitTorrent protocol favors peers with large upload bandwidth. Certain peers may experience poor download performance due to the disparity between users' upload/download bandwidth. The major objective of this study is to improve the download performance of BitTorrent users who have limited upload bandwidth. To achieve this goal, a modified peer selection algorithm and a cloud assisted P2P network system is proposed in this study. In this system, we dynamically create additional peers on cloud that are dedicated to boost the download speed of the requested user.

Contents

Abstract i
List of Figures iv
1 Introduction 1
2 Background 3
2.1 Client-server file distribution 3
2.2 Peer-to-Peer file distribution 4
2.3 BitTorrent overview 4
2.3.1 Framework of BitTorrent 5
2.3.2 BitTorrent protocols 5
2.3.2.1 Piece selection – rarest first algorithm 5
2.3.2.2 Peer selection: choking algorithm and optimistic unchoking 6
2.3 Related studies of BitTorrent-Like content delievery 8
2.5. Cloud Computing technology 9
2.6 AMAZON EC2 and Proposed Cloud Assisted Peer-to-Peer System 10
3 Implementation 12
3.1 Implementation of Classic BitTorrent' choking/unchoking algorithm 12
3.1.1 A flaw of Ctorrent 12
3.1.2 Root cause of the flaw 13
3.1.3 Fixing the choking/unchoking algorithm in Ctorrent 13
3.1.3.1 Data structure 13
3.1.3.2 Sorting process 14
3.1.3.3 Implementation of Optimistic Unchoking 17
3.2 Modification of BitTorrent Protocol choking/unchoking algorithm. 19
4 Results 21
4.1 Baseline tests 21
4.1.1 Baseline test setup 21
4.1.2 Baseline test results 23
4.1.3 Baseline test summary 27
4.2 Modified choking/unchoking algorithm tests 28
4.2.1 9-peer test: 1 assisting peers with 1000 KB/s upload bandwidth 29
4.2.2 9-peer test: 1 assisting peer with 2000 KB/s upload bandwidth 31
4.2.3 9-peer test: 2 assisting peers, each has 1000 KB/s upload bandwidth 33

4.2.4 9-peer test: 2 assisting peers, each has 2000 KB/s upload bandwidth 34
4.2.5 9-peer test summary 35
4.3 11-peer and 14-peer tests summary 36
5 Conclusion and Future Work 39
References 40

List of Figures

Figure 2.1 Client-server model…… 3

Figure 2.2 Peer-to-peer model……..... 4

Figure 2.3 Cloud assisted peer-to-peer system……... 11

Figure 3.1 Peer 9 who had 1KB/s upload speed is the fastest peer in the swarm…….. 12

Figure 3.2 Conceptual implementation of a peer list…….. 14

Figure 3.3 Overall steps of selecting peers to be unchoked. ……..... 15

Figure 3.4 pseudocode of checking download speed sorted peer list……. 16

Figure 3.5 Pseudocode of optimistic unchoking algorithm……..... 18

Figure 3.6 Assisted peer becomes unchoker directly……..…..... 19

Figure 4.1 Peer 1 has 1 KB/S upload bandwidth, Peer 2 – Peer 9 have 1000 KB/S Upload bandwidth …… 23

Figure 4.2 Peer 1 – Peer 9 have 1000 KB/S upload bandwidth……. 24

Figure 4.3 Peer 1 has 1 KB/S upload bandwidth, Peer 2 – Peer 11 have 1000 KB/S Upload bandwidth……. 25

Figure 4.4 Peer 1 – Peer 11 have 1000 KB/S upload bandwidth……... 25

Figure 4.5 Peer 1 has 1 KB/S upload bandwidth, Peer 2 – Peer 14 have 1000 KB/S upload bandwidth…… 26

Figure 4.6 Peer 1 – Peer 14 have 1000 KB/S upload bandwidth……... 26

Figure 4.7 Baseline test summary……..... 27

Figure 4.8 Peer 1 with 1 KB/S upload bandwidth, Peer 2 is the assisting with 1000 KB/S upload bandwidth, other peers (Peer 3 – Peer 9) also have 1000 KB/S upload bandwidth…….. 29

Figure 4.9 Peer 1's download speed with 1 assisting peer with 1000 KB/S upload bandwidth…….……. 30

Figure 4.10 Distribution of Peer 2's total upload……..... 30

Figure 4.11 Distribution of Seeder's total upload……..... 31

Figure 4.12 Peers' CPU utilization…… 31

Figure 4.13, 9-peer test peer 2 as assisting peer with 2000 KB/S upload bandwidth…… 31

Figure 4.14 Peer 1's download speed with 1 assisting peer with 2000 KB/S upload bandwidth……..... 32

Figure 4.15 9-peer test peer 2 and peer 3 as assisting peer with 1000 KB/S upload bandwidth..... 33

Figure 4.16 Peer 1's download speed with 2 assisting peers with 1000 KB/S Upload bandwidth ……..…….. 33

Figure 4.17 9-peer test peer 2 and peer 3 as assisting peer with 2000 KB/S Upload bandwidth…….……..... 34

Figure 4.18 Peer 1's download speed with 2 assisting peers with 2000 KB/S Upload bandwidth……. 34

Figure 4.19 9-peer test summary…… 35

Figure 4.20 11-peer test summary.…… 36

Figure 4.21 14-peer test summary.…… 37

Figure 4.22 Peer 1's download speed improvement in 11-peer and 14-peer test.……. 38

1 Introduction

As one of the most common peer-to-peer distributed application, BitTorrent helps people share their digital content cross the internet with significantly improved download speed. Since peers in a BitTorrent network function as both a server and a client, this high degree of decentralization solved the bottlenecking as well as the single-point-of-failure problem that come with the server-client system. However, not every peer could fully enjoy the benefit of joining a BitTorrent network. First, due to the nature of the BitTorrent protocol, peers who have higher upload bandwidth will be rewarded with higher download speed as they contribute to the network at higher rate [1]. This incentive-based mechanism leaves those peers who have limited uploading bandwidth have less chance to be selected and helped by other peers. Second, in real world, it is quite common that a peer may not take advantage of its high downloading capacity due to the disparity between its uploading and downloading bandwidth. [2] Third, simply due to insufficient number of peers, a peer may experience low downloading speed when the BitTorrent swarm is approaching the end of its lifespan.

To address these problems without introducing major changes to the classic BitTorrent Protocol [3] and to existing BitTorrent infrastructures, we propose a solution that designed to improve the downloading speed of peers with limited uploading bandwidth. In our proposed system, we incorporate cloud computing technology with the BitTorrent file distribution network. The on-demand nature of cloud computing technology also makes this solution highly applicable.

The hypothesis of our solution is that in addition to the peers that are chosen by the classic BitTorrent protocol, creating peers that are designed to help a specific user on a cloud platform can improve user's download performance.

This modified BitTorrent protocol along with cloud computing technology would address the inherent Quality of the Service [4] problem of the traditional BitTorrent network, and the proposed system keeps minimal changes on existing BitTorrent system, making it highly practical to deploy such a system in real world. This system also extends cloud computing service to the field of P2P networks. The traditional P2P file distribution system

could have improved overall performance meanwhile enjoying the benefit of the high elasticity provided by cloud computing technology [5]. The broader impact of the project is its potential far reaching benefit on increasing the revenue of the various companies that have businesses based on their BitTorrent-like protocol implementations and cloud service providers. *"More than 170 million people use BitTorrent product every month and its protocols move as much as 40% of the world's Internet traffic on a daily basis."* [6]. For such a large userbase, adoption of this new BitTorrent protocol will benefit both users and service providers such that it helps build a healthier and more profitable file-distribution ecosystem.

2 Background

In this work, we try to address a real-world problem – when a user who intended to download content using BitTorrent from the internet, e.g. a movie. However, this user may suffer from the low download performance due to its limited upload bandwidth. To address this problem, we investigated the choking/unchoking algorithm of the reference P2P protocol and modified this algorithm to improve the download performance of in a BitTorrent-like network. The fundamental idea of this work is to intentionally add peers to a peer-to-peer network, which are designed to help a specific user, so that the user download performance could be improved by continuously receiving data sent by these added peers. This chapter took an overview on traditional client-server file distribution architecture, peer-to-peer architecture, introduced the details of BitTorrent protocol, and proposed a system to address the forehead mentioned problem.

2.1 Client-server file distribution

Based on traditional client-server architecture [7], the file to be shared is being hosted on server side. The upload cost is cumulated when the number of users increases. However, when the upload bandwidth of the server is saturated as the number of users reached to certain point, further increment on user number will introduce performance issues. From the standpoint of users, they may experience a slower download speed than they would have when the server has not reach its upload speed limits.

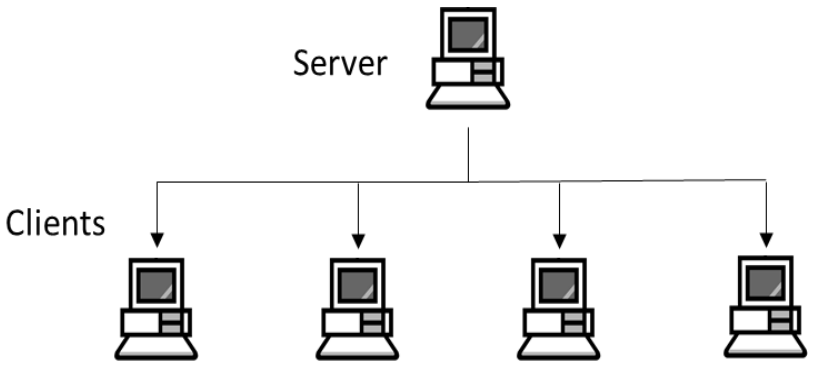

Figure 2.1 Client-server model

2.2 Peer-to-Peer file distribution

A peer-to-peer architecture [8] is a distributed architecture without the use of a centralized system. In this architecture, multiple users who are downloading the same file, for example, are connected to each other. This enable data exchange among users which largely reduced the upload responsibility that would place on the server in a client-server architecture. In effect, every connected user is a server and a client at once.

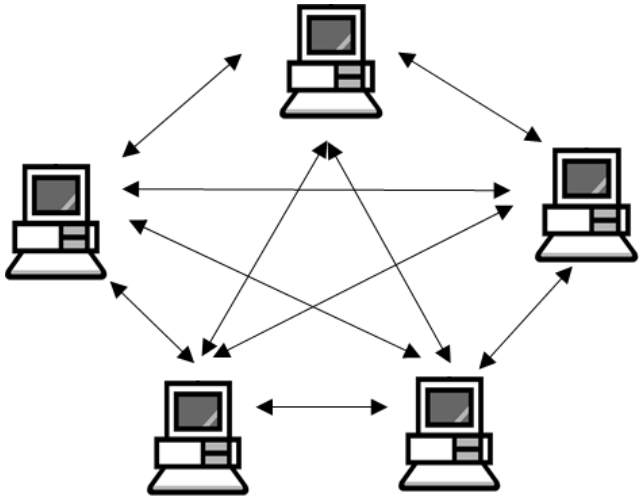

Figure 2.2 Peer-to-peer model

2.3 BitTorrent overview

BitTorrent was introduced in 2001, it is one of the most common applications that being used for large file distribution over the internet. The success of the BitTorrent is based on the fact that BitTorrent protocol well utilizes the p2p relationship among multiple users, which can significantly reduce work load that would be placed on the servers and can improve the download performances of users. Since a p2p network is highly scalable and self-organized [9] [10], as the number of users increases, the users will not suffer from any performance degradation that would be introduced in a client-server network, instead, they may experience an improvement on their download rates.

2.3.1 Framework of BitTorrent

Piece/Chunk: In BitTorrent, a file to be shared is divided into multiple fixed-size pieces. Any piece of a file can be exchanged among users. Once a peer acquires all the pieces, the file can be reassembled.

Seeder: A user who has the complete set of pieces/chunks of the file being shared. Users who completed the download and possesses all the pieces of that can file can change their roles into seeders. The owner of the original file is the first seeder.

Swarm: Group of peers and/or seeders who are downloading or uploading the same file.

Torrent file: A static file with the extension .torrent. Torrent file contains the metadata of the file to be shared, including the file name, length, hash list of all the pieces and the URL of trackers.

Tracker: A tracker is responsible for helping peers find each other. It maintains a list of peers that is currently downloading the file. When a peer joins a swarm, it first sends its own information to the tracker about what file it is going to download, and the tracker would respond with a list of peers. Then the new comer could use this information to connect to each other.

2.3.2 BitTorrent protocols

In BitTorrent, the file being distributed is divided into pieces and also peers are distributed over the internet. It requires effective piece selection and peer selection algorithms to delivery users efficient download performance.

2.3.2.1 Piece selection – rarest first algorithm

To make a file available to a swarm, it requires the first seeder, the original owner of the file, uploads all the pieces of the file to the swarm. This means that the total number of unique pieces all other peers have is same as the number of pieces which the file was divided into. Then, if the seeder leaves the swarm, other peers could acquire the complete file through exchanging the pieces among each other. In order to keep the file available as

long as possible in the swarm, then maximize the chance of acquiring the complete file for each peer, any peer should always consider downloading the piece that has the least number of existence in the swarm. This tactic is referred as rarest first piece selection. By implementing this algorithm in BitTorrent, it brings benefits to the peers from following two aspects:

- If the peer connected to a seeder, this algorithm could help seeder to send out all the pieces more efficiently, because redundant downloads increase the time it would take to send the complete file out. Thing gets worse if every peer that connected to the seeder downloaded pieces sequentially, then the purpose of using p2p network would be completely defeated, since every peer doesn't have the piece other peer want, so data exchange between peers cannot happen.

- This algorithm leaves more common pieces for later and the piece being downloaded would attract more interest from other peers, so that peers would be more motivated to exchange data.

2.3.2.2 Peer selection: choking algorithm and optimistic unchoking

The peer selection algorithms mainly consist of two parts, the choking algorithm and the optimistic unchoking algorithm. In the context of BitTorrent, choking conceptually means a peer refuses to upload its piece to others. However, in terms of implementation of BitTorrent, any peer at its progress of downloading will always unchoke a fixed number of peers.

Choking/unchoking Algorithm

Making decision of which peers to unchoke is based on a tit-for-tat strategy. In the context of BitTorrent protocol, tit-for-tat strategy means that when a peer is asked to upload its peers to other peers. This peer would first evaluate the amount of data that previously received from those requesting peers, then making the decision of whether to help other peers or not. For example, peer A would be willing to upload the piece requested by other peers only if the requesting peers already helped A on its own downloading, then peer A

would upload the requested pieces as a reciprocity. Tit-for-tat strategy helps maximize downloading speed of a peer, by evaluating which peers have had contributed the most to its own downloading. This can be figuratively understood as a peer tries to find its best friends among all the friends it has.

At implementation level, a peer keeps tracking the list of other peers it has downloaded pieces from and calculates the download speed of each connection for the last 20 seconds, afterward, this peer chooses, by default, the top 4 peers from which this peer had the most downloading speed and upload the requested pieces to these 4 peers, while other peers remain choked. For a seeder, the choking algorithm works differently, because the seeder has the complete file and no longer needs to download any piece from other peers. So, choices of unchoked peers is based on the criteria that to which peers that seeder can upload the piece as fast as possible.

However, the loophole here is that for a newcomer of a swarm who does not have any piece at hand to upload to any other peer will never receive any help from other peers. This prevents it from being unchoked by any other peer. Furthermore, it is also likely that a peer reciprocates to currently connected peers for a long period of time, which reduces a peer's chance of finding better download speed from the peers that remain unchoked

Optimistic unchoking

Optimistic unchoking algorithm is designed to address the above issues. In addition to the tit-for-tat strategy, a peer A would randomly choose to unchoke another peer B regardless of the download rate from this peer B and upload whichever piece peer B wants for every 30 seconds. Optimistic unchoking can be helpful to the new peers in the swarm, and the peer who performs optimistic unchoking also increases its chance of getting better download speed from those previously choked peers.

2.3 Related studies of BitTorrent-Like content delievery

Dongyu Qiu and R. Srikant proposed a fluid model [11] for BitTorrent-like networks. They modeled peer arrival and departure behaviors, evaluated the download speed of peers and seeders in various settings and measured the performance of the BitTorrent-like networks and found out mathematical description of the impact of these variables.

The authors of "*Clustering and Sharing Incentives in BitTorrent System"* [12] found that the classic design of BitTorrent protocol would introduce clustering effect, by which peers are more likely to connect to other peers who had similar bandwidth capacity. The authors argue that the clustering effect is beneficial to data sharing among peers, because this effect would reward peers who had better upload bandwidth utilization.

However, in another study [13], The authors suggested that the amount of rewards received by peers who has higher upload bandwidth does not increase proportionally to their higher upload utilization.

Amita Ajith Kamath, Chirag Jamadagni, K. Chandrasekaran propsed VirtTorrent [16], which improves data distribution in a cloud computing setting. They optimized the communications among virtual machine in a cloud platform and reduced the data sharing latency among instances.

Concept of sequential progress was proposed in a study [14]. Authors of this studied proposed a new piece selection algorithm which would perform better in media streaming applications, comparing to the original BitTorrent protocol. They claimed that media streaming startup delay would be significantly reduced by their first-come-first-serve piece selection algorithm, whereas, Rarest-First algorithm in BitTorrent protocol performs poorly in streaming scenario.

Similarly, by applying BitTorrent peer-to-peer technology, C. Lo and Y. Su [15] proposed a system which could facilitate content delivery for Video-on-Demand services. They successfully redirected the traffic from the VoD server to a peer-to-peer network. In this way, the workload on the VoD server can be reduced while maintaining users' experience.

2.5. Cloud Computing technology

With the success of Internet and tremendous advance in processing and storage technologies, a new computing model called Cloud Computing has gained its popularity significantly. The term Cloud computing has been intensively used as a marketing term which obfuscate its original technical meaning. In fact, the main ideal behind Cloud Computing is straight forward. That is to provide computing facilities to publics. Conceptually, the practice of cloud computing is analogous to the practice of providing electricity to every household from a power plant. The definition of cloud computing Cloud computing from the National Institute of Standards and Technology is –

A model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g. Networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [17].

According to this definition, the traditional role of a service provider in a Client-Server model has changed in a Cloud Computing environment. First, the physical computation facilities are deployed and maintained by infrastructure providers and they provide these computation facilities to public as a service in an on-demand fashion. Second, the service providers, as users of infrastructure providers, rent those computation facilities and running hosting their own applications, which can be delivered to the end users.

Cloud computing brings benefits to its users from following two aspects:

1) Ease of use/cost effective [18]. For service providers, the amount of computing power and resources they need usually significantly larger than the amount needed for personal usage, then data storage, power consumption and heat dissipation issues, which are hardly considered when using personal computer, may become huge challenges. With Cloud Computing, the burden of maintaining local computation facilities is placed on the infrastructure providers. This would save service providers from deploying and maintaining physical computation facilities and lowering their operating cost.

2) Resource elasticity [19]. One major disadvantage of hosting physical facilities is its lack of scalability. If a company wants to accommodate the maximum number of users they would have, then they would have to allocate computing resources at its peak load. However, users' demand waxes and wines, a maximum fixed-size deployment will introduce unnecessary cost to the company. One the other hand, using their virtualization techniques, infrastructure providers can deallocate and reallocate their computing resources rapidly so that service providers could dynamically configure their service capacity according to their users' demand.

2.6 AMAZON EC2 and Proposed Cloud Assisted Peer-to-Peer System

AMAZON EC2 is an Infrastructure as a Service (IaaS) cloud computing platform [20]. EC2 features fast instances initialization as well as a high degree of customization. Its users can configure the computing capabilities and resources based on their personal or business need. Its "you-pay-as-you-go" billing model is highly cost effective. Our proposed system is implemented on Amazon EC2 platform, taking advantages of its effortless IP address-Instance association and disassociation. Instances provided by Amazon EC2 varies by their core count, storage, memory and network throughput. Different applications can benefit from this great amount of choices. Our proposed system utilizes its Linux t2. micro instances and its computing capacity is well suited in our use case. As one of the most cost-effective type of virtual machine, evidence of micro instance sufficiency is proven in our system performance measurement

Since cloud computing technology provide computing resources to the public based on its hardware virtualization technique. It features fast resource provision and elastic service scalability. This great flexibility offered by cloud computing perfectly matches with the dynamic of the P2P network. In this study, we create additional peers on Amazon EC2 platform that would help a user regardless of the amount of data this user uploaded to those created peers on the cloud. In this manner, the user can enjoy increased downloading speed without increase its uploading traffic.

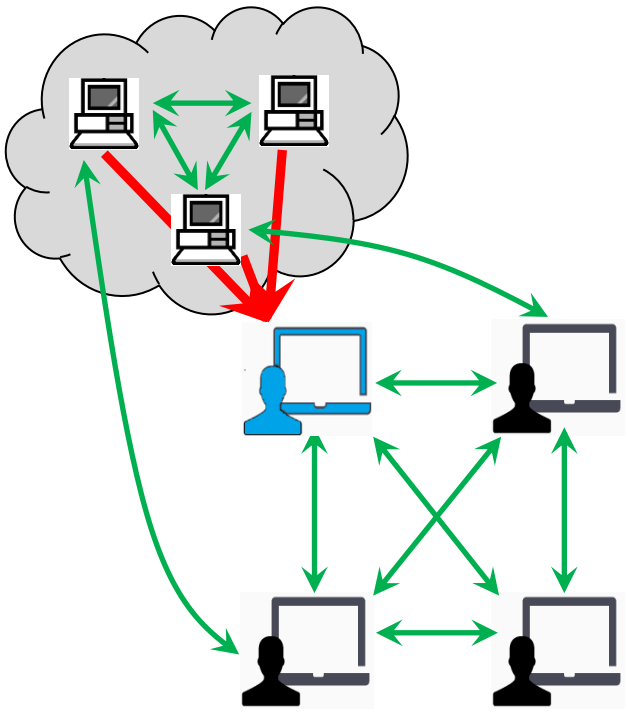

Figure 2.3 Cloud assisted peer-to-peer system.

The cloud assisted peer-to-peer system proposed in this work is conceptually shown in Figure 2.3, where the user/peer being assisted is shown in blue icon and the black icons shows other users in a traditional peer-to-peer network. Give the on-demand nature of cloud computing technology, it provides a natural solution to improve the under-performance peer, as additional peers on the cloud join user's network at any requested time and could also be terminated whenever the user feel comfortable to.

In this study, we implemented a modified choking/unchoking algorithm that running on the peers created on the cloud and simulated this system on Amazon EC 2 platform. We evaluated the performance of this system in terms of how effectively of the algorithm could contributed its upload bandwidth to the requested peer.

3 Implementation

3.1 Implementation of Classic BitTorrent' choking/unchoking algorithm

3.1.1 A flaw of Ctorrent

The baseline code of this work is based on a widely used BitTorrent protocol implementation called Ctorrent. It is registered application in official repository of Ubuntu Linux operating system. Given its large popularity, easy accessibility and open sourced nature, we decided to use this application as our baseline benchmark.

However, after extensive test and investigation, I found that the author of Ctorrent implemented BitTorrent protocol in a way that would unfairly favors peers with much less uploading bandwidth, which defeats the idea of "discouraging free riders" [20] in BitTorrent protocol. Following is an example of this unfairness. In this swarm, there are nine (peer1 to peer9) peers and one seeder. From peer1 to peer8, they all had 1000 KB/S upload bandwidth, peer 9 only had an upload bandwidth of 1 KB/S. The seeder had an upload bandwidth of 1000 KB/S. All 9 peers joined the swarm at the same time with zero downloaded data. The download speed of peer 9 was expected to be much slower than the other 8 peers. However, the result shown a completed opposite outcome.

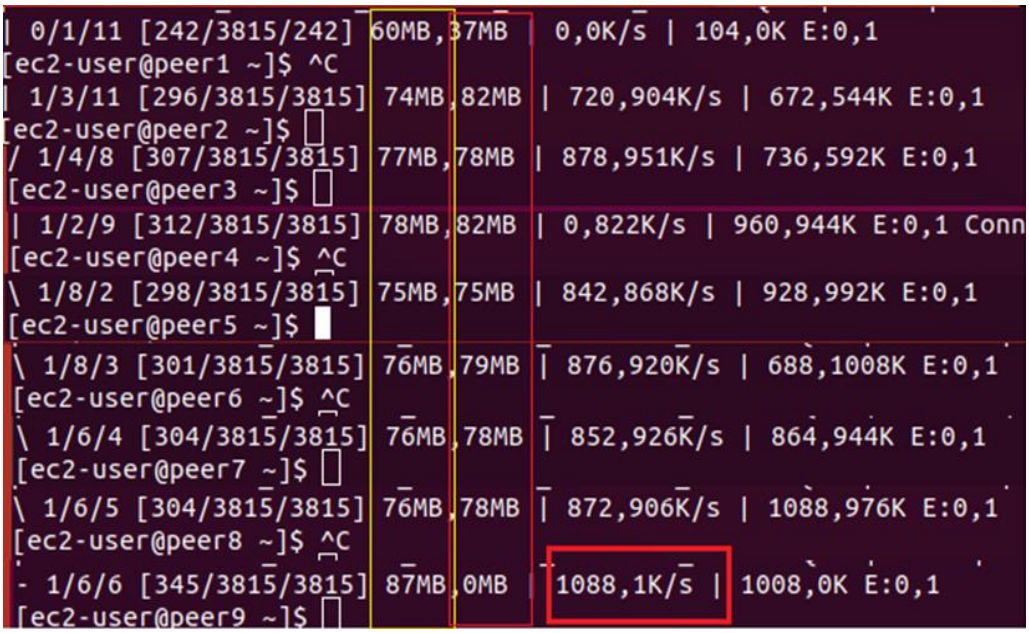

Figure 3.1 Peer 9 who had 1KB/s upload speed is the fastest peer in the swarm.

The yellow rectangular area indicates the total download each peer had, and the red rectangular area indicates the total upload each peer had. Cleary, peer 9 has most downloaded data of 87 MB and had a 1088 KB/S download speed with an upload speed of 1 KB/S.

3.1.2 Root cause of the flaw

Root cause of the flaw mentioned about it that the author of Ctorrent implemented its optimistic unchoking algorithm in a biased way. The author maintained a linked list data structure that has all the peers as its nodes. After traverse through the list, it found the fastest three peers and one slowest peer. However, instead of choosing its optimistic unchoker in a more random manner, the author always considers the slowest peer its potential optimistically unchoked peer.

This unchoking algorithm gives much more opportunity to peers that with little upload data to become optimistically unchoked peer. So, if the peer intentionally decided not to reciprocate or had very limited uploading bandwidth, the peer would not be punished by this algorithm. This would put people who are running this client in a unfair position that their uploading bandwidth is exploited by other peers who are not able or not willing to reciprocate.

3.1.3 Fixing the choking/unchoking algorithm in Ctorrent

3.1.3.1 Data structure

To implement the choking/unchoking algorithm effectively, it is crucial to maintain the information of each peer properly and order them by their speed efficiently. In this case, linked list structure comes naturally here, consider each peer as its node, peers 'joining and leaving can be easily tracked by simply adding or removing a node in the list. The following figure shows what make of this peer list.

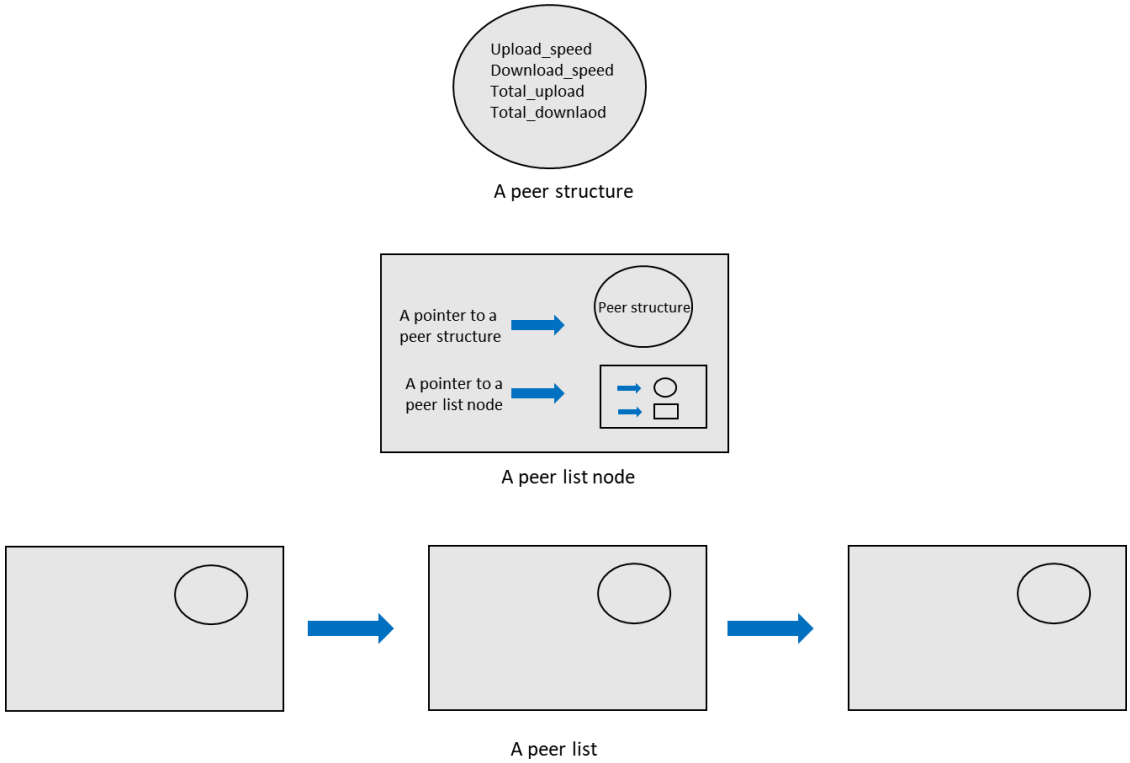

Figure 3.2 Conceptual implementation of a peer list

3.1.3.2 Sorting process

To implement the choking/un-choking algorithm correctly, choosing those fast peers is the key procedure. To elaborate my implementation of this algorithm, it is necessary to introduce the main components in my algorithm. Please note that for the ease of explanation, I assume there are four allowed peers (as specified in BitTorrent Protocol) to download. In the real algorithm implementation, the number of allowed peers can be configured by user.

- **Peer list**. A list of connected peers, unsorted, peers are connected based on their arrival time.
- **Interested-Peerlist**. A list of peers that interested in my data, unsorted.
- **Sorted_Peerlist.** This is the peerlist that is sorted by peers download speed from the fastest to the slowest. This list, if necessary, will be further processed based on peers' total download and total upload.

- **Unchoker_containter**. This is an array of four pointers, the first three pointer point to three fastest peers and the last pointer points to the optimistically unchoked pointer.

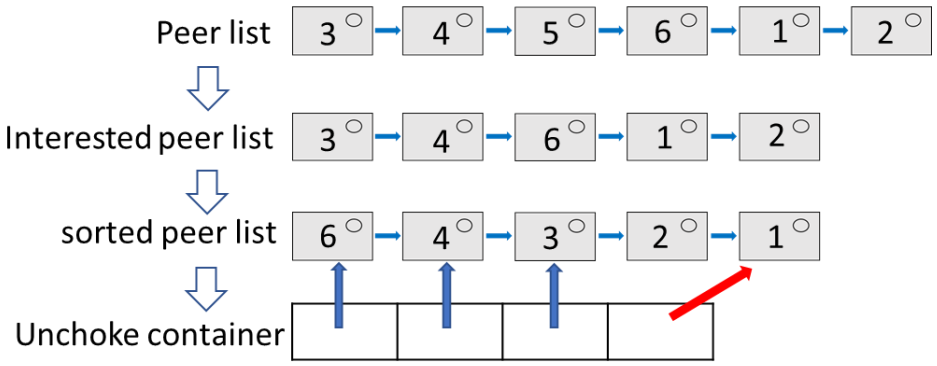

Figure 3.3 Overall steps of selecting peers to be unchoked. The numbers in the peer list node indicates the speed ranking of each peer if sorted. The red arrow represents the pointer points to optimistically unchoked peer.

The challenging part of this implementation is the priority sorting algorithm. Since a peer may the same download speed from other peers, to better utilize finite upload bandwidth, a peer chooses its unchoked peer in follow priority

Download_Speed > Total_Download > Total_Upload

The rationale of this priority is

- Sort the peer list by their download rate, then peer will reciprocate to those peers from whom it has best download rate,
- If from two peers, this peer has the same download speed, reciprocate to the peer from whom it has more download data.
- If there are two peers that from whom I have the same download speed and total downloaded date, reciprocate to the peer that it has less upload data to.

```
For(index<number of unchoker)
{
    Traverse through the download speed sorted peer list
}
  // now we stopped at the boundary_node

if (boundary_node->peer->total_download  == boundary_node->next->peer->total_download)
  // equal download rate happened at the boundary
 {
       //Log the total_download value at the boundary
        targe_value = boundary_node->peer->total_download
       //check how many same total download afterwar
       Int after =0
      traverser_1 = boundary_node;
      while (traverser_1->next is not null && traverser_1->next->peer->total_download
            == targe_value)
      {
         after++;
         traverser_1=traverser_1->next;
       }
       //check how many same downloader ahead
       before = 4;     // assume they all have the same download from the 1st peer
       traverser_2= first_node;

       while (traverser_2->peer->total_download!=targe_value  && before>1)
       // if we need to do this check, it means at least 1 peer within unchokers boudary has
       // the same dl speed as the it of peer right after the unchokers boudary;
       {
          traverser_2=traverser_2->next;
          before--;
       }
       traverser_3 = first_node;
       for (int i =0; i<(4 -before);i++)
       {
          traverser_3 = traverser_3->next;
       }
       upload_sort_start =traverser_3;

       traverser_4 = boundary_node;
       for (int i =0; i<after;i++)
       {
          traverser_4 = traverser_4->next;
       }
     upload_sort_end=traverser_4

    check_total_upload_sorted_list(upload_sort_start, upload_sort_end);

  }
```

Figure 3.4 pseudocode of checking download speed sorted peer list

To improve the efficiency of sorting on peers, instead of finding out all those peers who have the same download rate, my implementation first exams the peers near the boundary

(the 4th peer and 5th peer) in sorted peer list to decide if further sorting is needed. If they have the same download rate, then the algorithm will find out how many consecutively connected peers in the list crossing the boundary that have the same download rate. Then two pointers, pointing to the start and the end of these consecutively connected peers respectively, will be passed to next step processing (sort peers by their total download). Afterward, this checking will be applied again to the download_speed/total_download sorted peer list. Eventually, we will get a list that sorted by their download_speed, total_download and total_upload. The pseudocode of this algorithm is given as Figure 3.4.

3.1.3.3 Implementation of Optimistic Unchoking

Optimistic unchoking, as stated in BitTorrent protocol, is a process that to check currently unused peers with the hope that from those unused peer, it may connect to faster uploading peers than the peers it is currently connected to.

After applying previous sorting algorithm, we acquired a peer list that sorted based on BitTorrent protocol requirement and given this list, we are able to find out which peer should be choked or unchoked. In my implementation, I use an array of four pointers to access those peers that should be unchoked.

Since BitTorrent protocol specifies that regular unchoking happens every 10 seconds and optimistic unchoking happens every 30 seconds. Therefore, before selecting peers from the sorted list, first, it is needed to check the timer to find out if it is time to do optimistic unchoking. If it is, then one pointer of the array of unchoker container should be reserved for next 30-second optimistically unchoked peer. The pseudocode of the optimistic unchoking algorithm is shown in Figure 3.5.

```
traverse through the sorted peer list until firs 3 unchoker container is full
 {
        If  (the peer is not the previously optimistically unchoked peer)
        {
                put it into one container
        }
        Else
        }
                Put the this peer (optimistically unchoked) to the last container.
        }
} // After, this step, the first 3 containers are guaranteed to be filled in with peers.
  // But the last (4th) container may or may not be filled.

if (the 4th container is not empty)  //previously optimistically unchoked peer is
found
{
        if (this is a regular unchoking)
        {
                All the container are filled, choke all the other peers that not in
                the container.
        }
        Else
        {
                chose another peer from the remaining of the list to be
                optimistically unchoked peer and put it the last container
        }
        choke all the other peers that not in the container.
}
Else  // the previously optimistically unchoked peer is not found after filling in the
      //first three container.
{
        if (this is a regular unchoking)
        {
                find the previous optimistically unchoked peer in the remaining
                list and put it to the last container

                choke all the other peers that not in the container.
        }
        Else
        {
                randomly chose another peer from the remain list to be
                optimistically unchoked peer and put it the last container
        }
        choke all the other peers that not in the container
  }
```

Figure 3.5 Pseudocode of optimistic unchoking algorithm

In the BitTorrent protocol, it states that "*new connections are three times as likely to start as the current optimistic unchoke as anywhere else in the rotation*.", in my implementation, if selected optimistically unchoked peer is the same as last round selected peer, then it only has 25% percent of chance to be considered as a final unchoker.

3.2 Modification of BitTorrent Protocol choking/unchoking algorithm.

The idea of assisting a peer is implemented by excluding the peer from the choking/unchoking algorithm. The peer running this modified algorithm would upload to the assisted peer whatever it wants, regardless of its download rate from the assisted peer. This means the assisted peer should remain unchoked all the time in its peer list.

Given the assumption that the peer being assisted has very limited uploading bandwidth, then the peer who keeps helping this slow peer would get little download from the assisted peer, as a result, if the assisting peer would have less download speed comparing to the other regular peer, therefore less downloaded content. Although the assisting peers are not interested in downloading the content as fast as possible, maintaining a decent load speed, in return, would also help maintain its ability to upload data to the assisted peer. This means it is better not to have the assisted peer occupy one of the four unchoker containers, and then the assisting peer would have one more unchoker than it would normally have.

Setting aside the assisted peer from other peers can be implemented at different stage of the choking/unchoking algorithm, the most straightforward and efficient way of doing so is to exclude the assisted peer from currently interested peer list, then the following sorting algorithm would not be applied to the assisted peer.

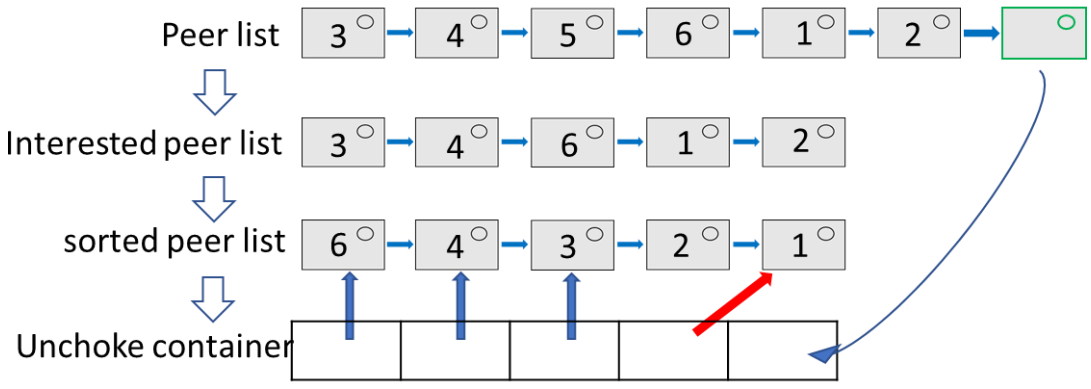

Figure 3.6 Assisted peer becomes unchoker directly

To differentiate the assisted peer from other peers, another Boolean attribute is added to the peer structure, this attribute is set to be true, when the first time the peer is added into the peer list.

The idea of this modified choking/unchoking algorithm to assist another peer in conceptually in shown in the Figure 3.6, where the green outline peer node is the peer to be assisted. This peer bypass all the regular choking/unchoking algorithm and the 5^{th} unchoker directly.

4 Results

Evaluating the performance of a choking/unchoking algorithm turned out to be surprisingly challenging, because of the dynamics of different swamps vary significantly. Even for the same swamp, its dynamic changes over time. Seeders 'upload bandwidth, peers' upload/download bandwidth, number of peers, download completion percentage of each peers and peer's arrival and departure would impact peers' download/upload performance. To acquire comparable results from run to run, we tested our modified choking/unchoking algorithm in a well-controlled setup and all the tests were conducted on Amazon EC2 cloud platform using their micro Linux instances.

4.1 Baseline tests

4.1.1 Baseline test setup

For this study, we are interested in how an upload bandwidth limited peer would perform in a resource limited swamp. To test proposed choking/unchoking algorithm, we also wish to maximize data exchanges among peers. With these requirements in mind, we setup our test environment as following:

File size: 1GB (1,000,000,000 Byte)

Number of Seeder: 1

Seeder upload bandwidth: 1000 KB/S

Number of peers: 9, 11, 14

Peers' upload bandwidth: 1000 KB/s

Peer arrival: Every peer joins the swamp at the same time with zero data on hands.

Peer departure: Not modeled, download/upload readings are measured when the first peer downloaded 500MB of data.

1) To simulate a "resource limited" swamp, we use 1 seeder to do uploading, and on each round of test, we clear out any available data on each peer and let them join

the swamp as a fresh start. In this manner, we ensure that, at the beginning of each round of test, there is only one complete copy of data in the swamp and, therefore, peers data exchanges are maximized.

2) To ensure test consistency, we allow the seeder to unchoke every peer in the swamp and distribute its upload bandwidth equally across all the peers. This is because we don't want any biased seeder's upload behavior to blend into our test results. Thus, peers' upload/download performance is primarily depending on how the choking/unchoking algorithm (data exchange) performs.

3) The reason we choose to use 1000 KB/S (8 Mbit/S) uploading is based on the statistical data of an average upload bandwidth of cable ISP in the US of 22.79 Mbit/s and 8.51 for mobiles [22]. Considering seeder usually would not seed with it full bandwidth (may reserve some for other activity), we empirically/experimentally choose 1000 KB/S as our upload bandwidth.

4) As to the reason for presenting performance results starting from 9 peers is that we observed a major (30%) performance drop on upload bandwidth limited peer when total peer number is greater than 8 in this test setup. With peer number of 5, the upload bandwidth limited peer would not be punished by not contributing data to others, since the number of peer to be unchoked is defined as 4 in the classic BitTorrent protocol.

5) Peer departure behavior is not modeled. This is because in a real-world situation, when a peer fully downloaded the data, how long it would keep seeding vastly varies among peers. To avoid this uncertainty, we decided to measure test upload/download readings at the time when the first peer in the swamp downloaded 500MB of data.

4.1.2 Baseline test results

To acquire a baseline that could be compared with the performance of proposed choking/unchoking algorithm, we investigated how upload bandwidth of a peer (shown as peer 1 in our test results) would impact its download speed in two scenarios -- normal bandwidth scenario and extremely limited bandwidth scenario. Each scenario was tested 5 times in 9, 11 and 14 peers swamp respectively.

The following figures (Figure 4.1 – Figure 4.6) show the averages download and upload speed of each peer in their tests.

- **Baseline test** – tested in extremely limited bandwidth normal bandwidth scenario where peer 1 has an upload bandwidth of 1KB/s, the rest of peers have same upload bandwidth of 1000 KB/s.
- **Reference test** – tested in normal bandwidth scenario where peer 1 has the same upload bandwidth, 1000 KB/s, as the rest of peers.

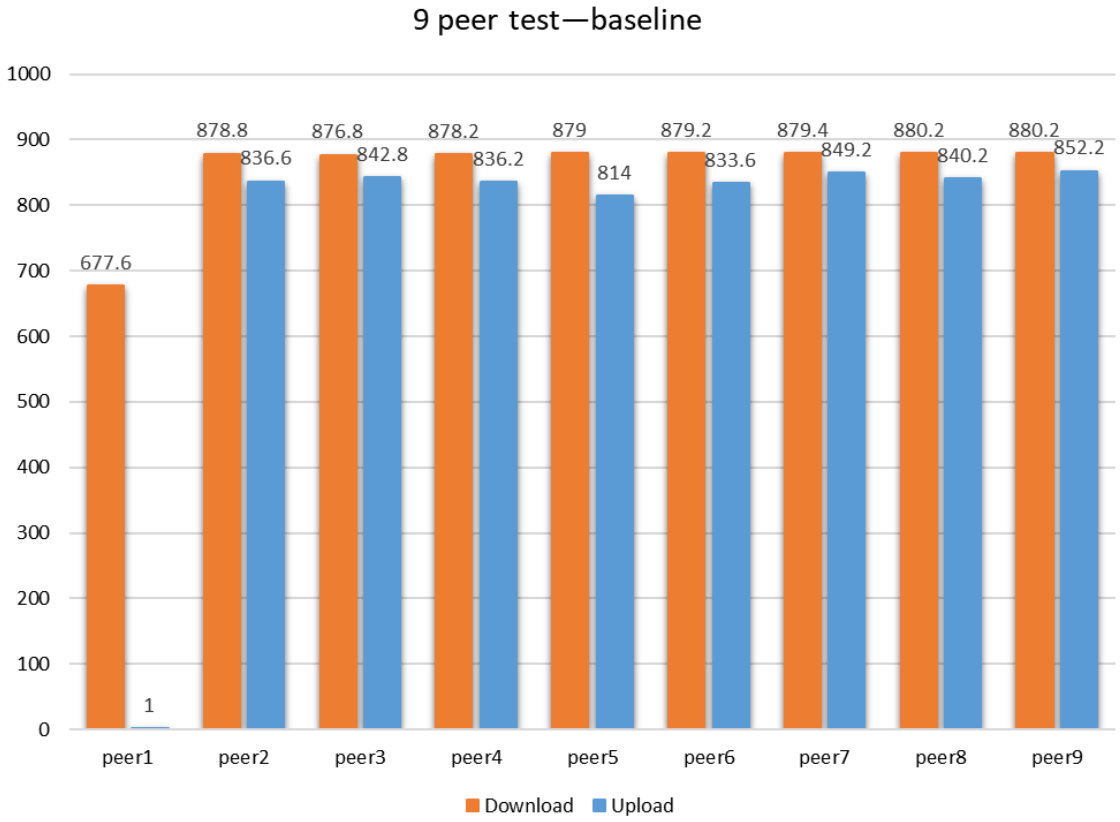

Figure 4.1, Peer 1 has 1 KB/S upload bandwidth, Peer 2 – Peer 9 have 1000 KB/S upload bandwidth

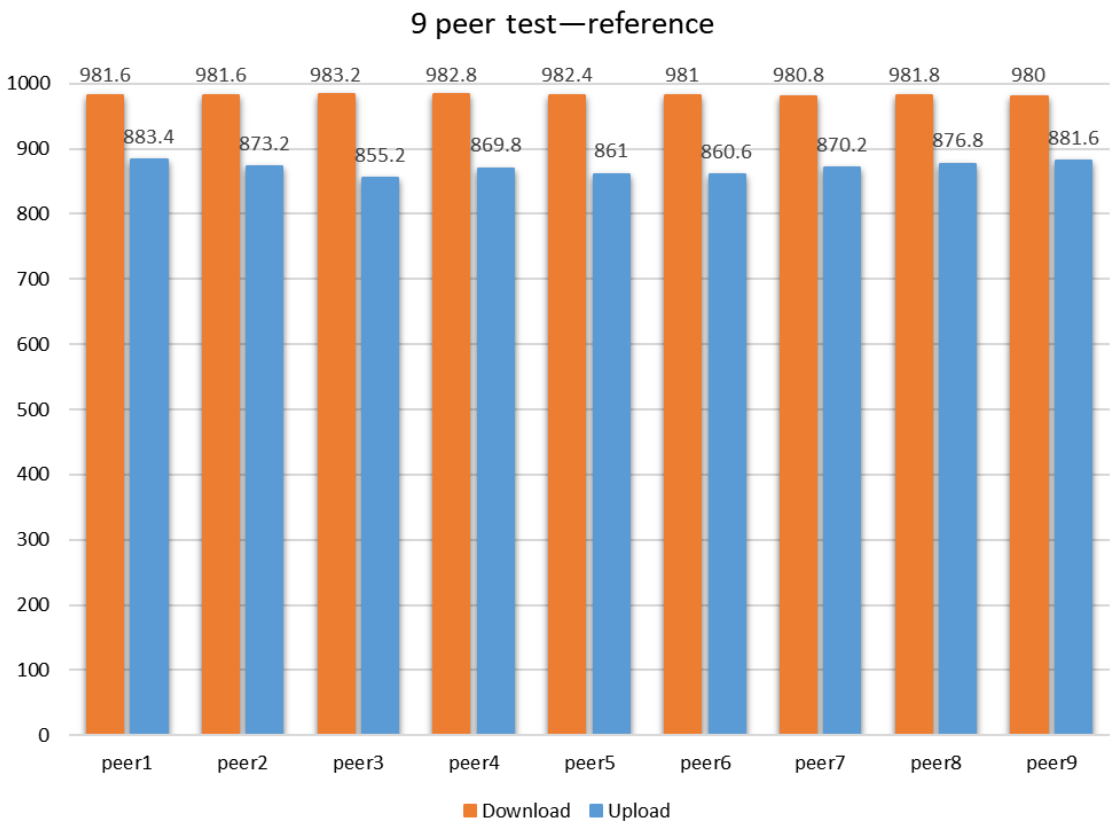

Figure 4.2, Peer 1 – Peer 9 have 1000 KB/S upload bandwidth

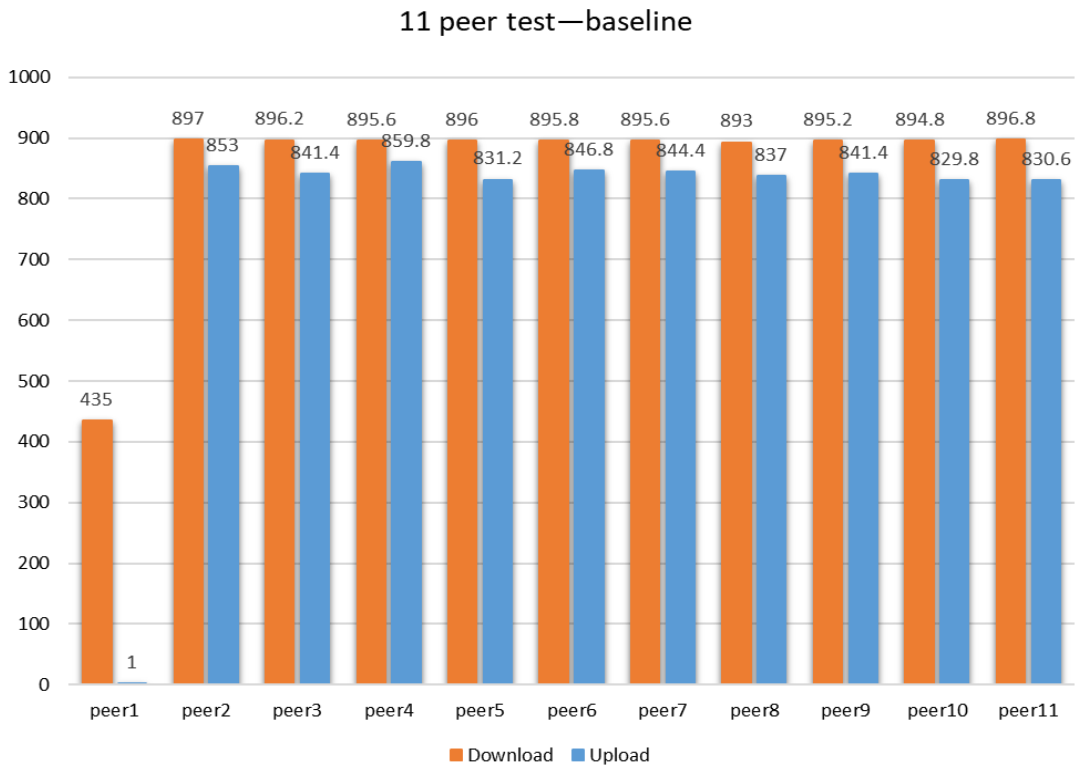

Figure 4.3, Peer 1 has 1 KB/S upload bandwidth, Peer 2 – Peer 11 have 1000 KB/S upload bandwidth

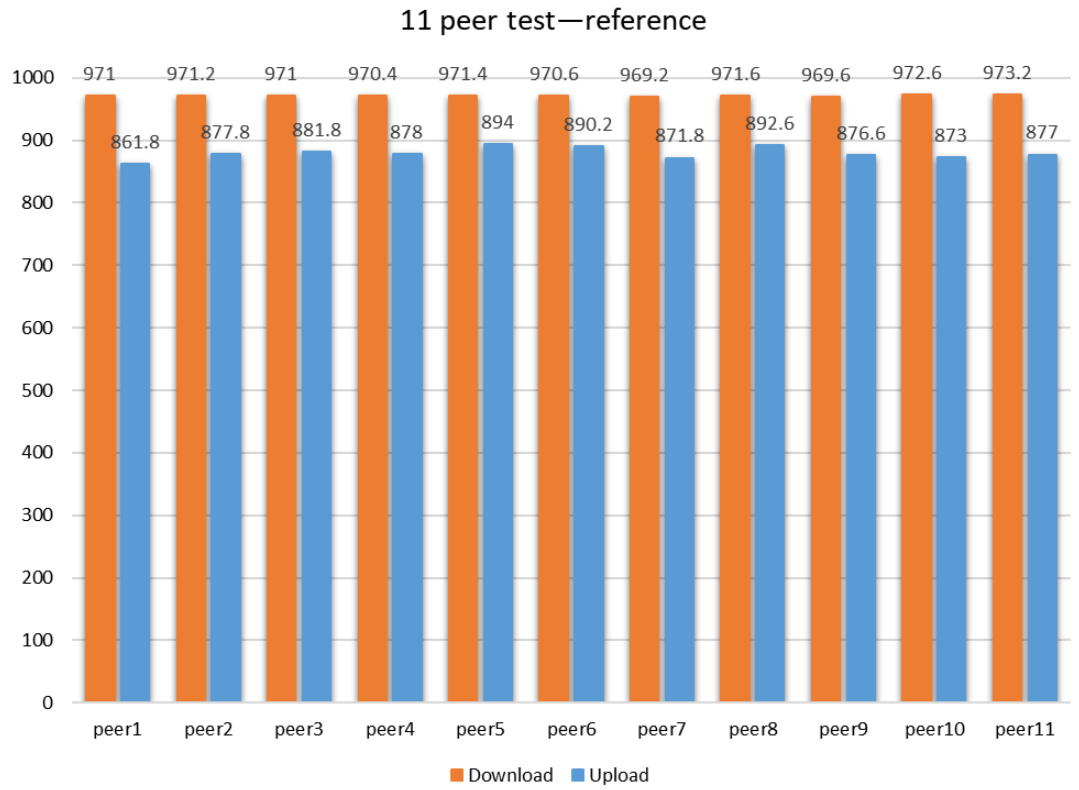

Figure 4.4, Peer 1 – Peer 11 have 1000 KB/S upload bandwidth

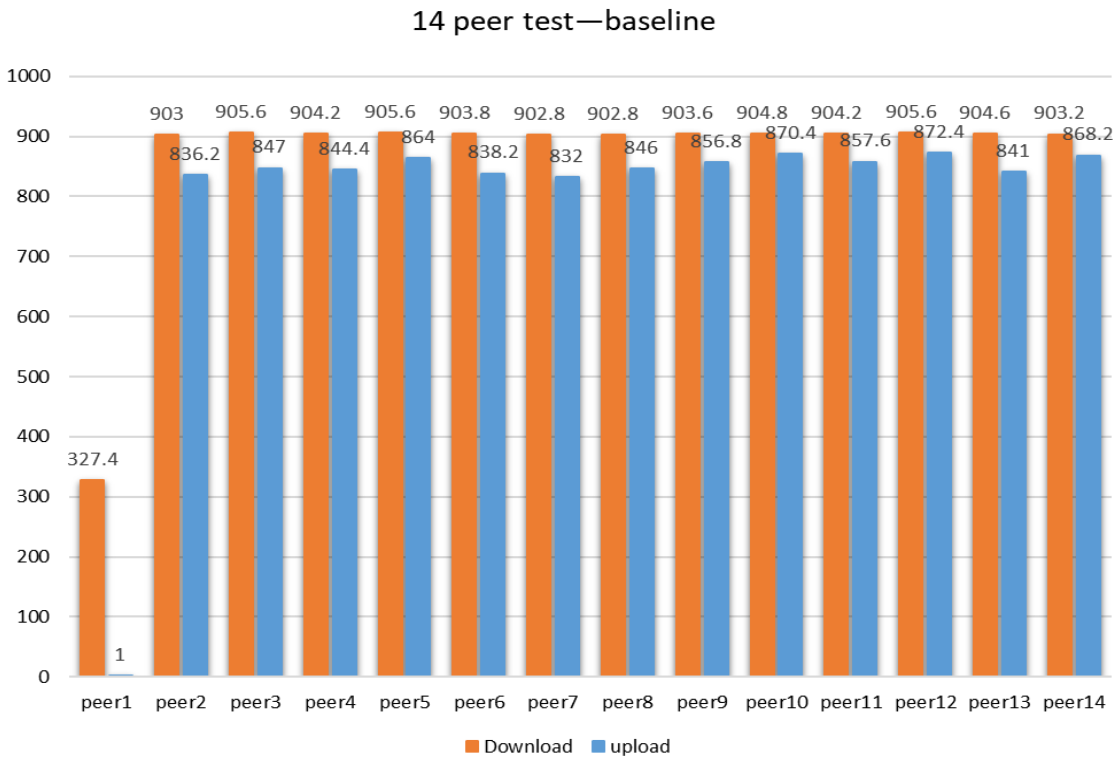

Figure 4.5, Peer 1 has 1 KB/S upload bandwidth, Peer 2 – Peer 14 have 1000 KB/S upload bandwidth

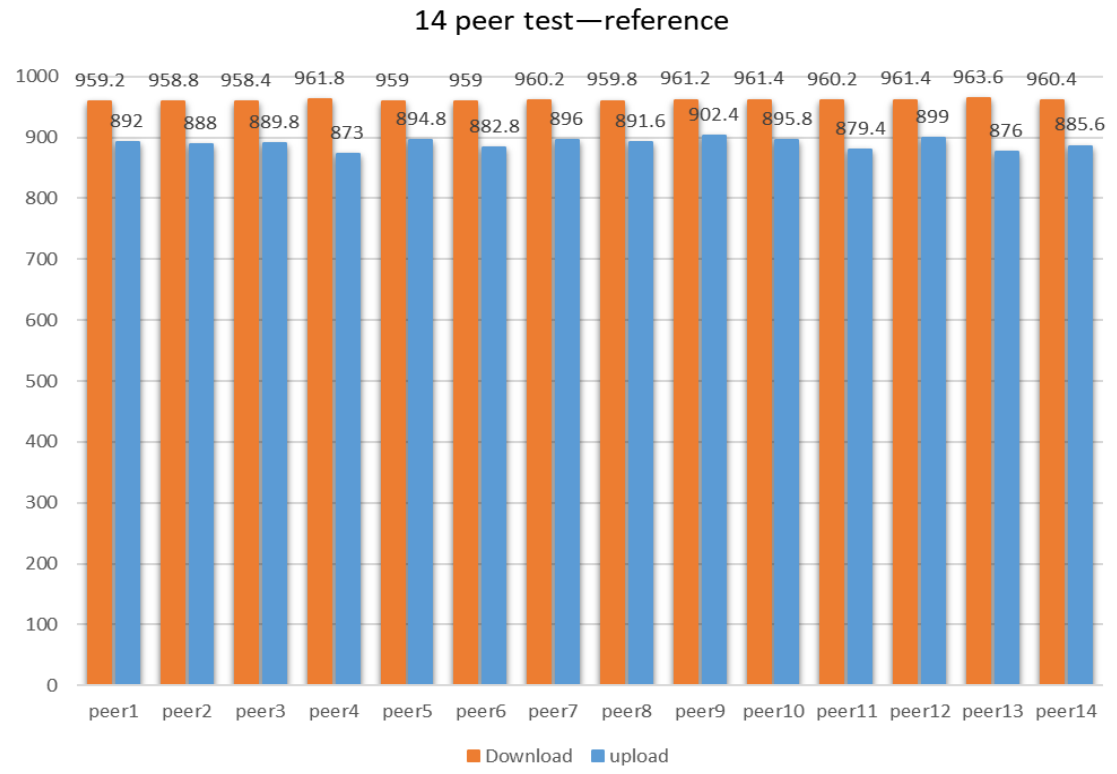

Figure 4.6, Peer 1 – Peer 14 have 1000 KB/S upload bandwidth

4.1.3 Baseline test summary

	Peer1 with 1 KB/S upload bandwidth		Peer1 with 1000 KB/S upload bandwidth		Slow down on Peer1's download speed	Slow down on other peers download speed
	Peer1 average download speed	Others average download speed	Peer1 average download speed	Others average download speed		
9 peer test	677.60	878.98	981.60	981.70	-30.97%	-10.46%
11 peer test	435.00	895.60	971.00	971.08	-55.20%	-7.77%
14 peer test	327.40	904.14	959.20	960.40	-65.87%	-5.86%

Figure 4.7, Baseline test summary

The results of baseline tests demonstrate that the effect of optimistic unchoking on "slow" peer becomes less significant with increasing number of peers. Since each peer would have more peers to select from, then "slow" peer would have less probability to be optimistically unchoked. Therefore, peer 1's download speed dropped significantly where the number of peers in a swamp increases.

Limited upload bandwidth not only slows down the peer, but also "slow" peer slows down the whole swamp. As shown in figure 4.7, In the 9-peer test, when peer 1 had 1KB/s upload bandwidth, other peers in the dropped their download speed about 10%. This is because peers download speed depends on data exchange among peers. Any peer who unchoked peer 1 would expect peer 1 to reciprocate afterwards. However, in every 1 KB/S test, peer 1 could hardly return meaningful amount of data to other peers, thus slow down the whole swamp. Fortunately, slow peer has less impact on other peers as swamp size grows.

When peer 1 had 1000 KB/S upload bandwidth, average download speed of other normal peers (who had 1000 KB/S upload bandwidth) still decreased with increasing swamp size. This is due to the whole swamp download speed is seeder-bounded when every peer joined the swamp at the same with zero available data. Seeder needs to upload some data to other peers before those peers could start data exchange. Since we keep seeder upload bandwidth as 1000 KB/S across all the tests, it would take more time for seeder to upload data to each peer when number of peers increases. The upload bandwidth is equally distributed among peers. Figures of this seeder behavior are shown in following test.

On the other hand, given peer 1's 1 KB/S upload bandwidth, average download speed of other normal peers (who had 1000 KB/S upload bandwidth) increases with increasing swamp size. This is the fact that when there are more peers in a swamp, the slow peer would have less chance to be unchoked by other peers, and other peers have more opportunity to get reciprocated data. Then slow peer's "free riding" behavior become less dominant.

Please note that the following discussion about the performance of modified choking/unchoking algorithm is based on these baseline test results. The reference speed mentioned hereafter refers to the result of normal bandwidth scenario where peer 1 has the same upload bandwidth, 1000 KB/s, as the rest of peers, and baseline speed refers to the results of extremely limited bandwidth normal bandwidth scenario where peer 1 has an upload bandwidth of 1KB/s.

4.2 Modified choking/unchoking algorithm tests

We investigated how effective it could be when we use modified choking/unchoking algorithm on other peers to help a slow peer. Following our baseline test, we kept peer1 as a slow peer with 1 KB/s upload bandwidth, and there are three main questions need to be answered in our tests:

Given current test setup,

1) How many peers should run the modified choking/unchoking algorithm to improve slow peer's download speed to a reasonable level?
2) Which approach would bring more improvement, increase number of assisting peers or increase the upload bandwidth of assisting peers?
3) How effective the modified choking/unchoking algorithm is when it comes to redirecting its upload bandwidth to the slow peer?

Our investigation on these three questions started with our 9-peer (and 1 seeder) tests. In our first round of tests, we started our test with one assisting peer in the swamp which had

the same upload bandwidth of 1000KB/s as other peers, then we doubled the bandwidth (2000 KB/s) of the assisting peer. In the second round of test, we applied 2 assisting peers which had 1000 KB/s upload bandwidth in the swamp, afterwards, we tested 2 assisting peers with 2000 KB/s upload bandwidth in the swamp.

4.2.1 9-peer test: 1 assisting peers with 1000 KB/s upload bandwidth

The following figure 4.8 shows the result average download and upload speed of all the peers in a 9-peer swamp, where peer 2 is the assisting peer.

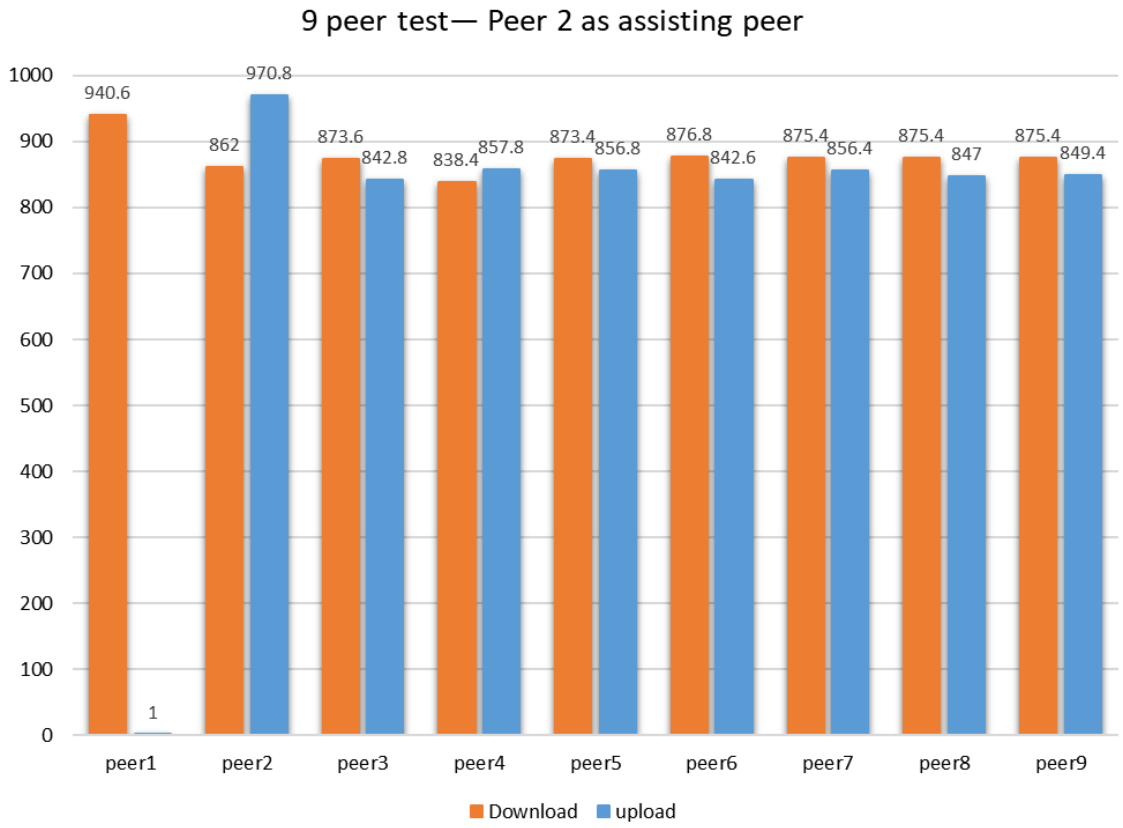

Figure 4.8, Peer 1 with 1 KB/S upload bandwidth, Peer 2 is the assisting with 1000 KB/S upload bandwidth, other peers (Peer 3 – Peer 9) also have 1000 KB/S upload bandwidth

	Without assisting peer	With 1 assisting peer 1000 KB/S upload bandwidth	Improvement over baseline speed	Compare against reference speed
Peer 1's average download speed (KB/S)	677.60	940.60	38.81%	95.82%

Figure 4.9 Peer 1's download speed with 1 assisting peer with 1000 KB/S upload bandwidth

As shown in figure 4.8 and figure 4.9, peer 1's average download speed increased by 38.81% and reached 95.82% of the reference speed. To confirm this improvement was introduced by the assisting peer (peer 2). We acquired the upload bandwidth distribution of peer 2. Figure 4.10 clearly shows that peer 2 effectively contributed most of its bandwidth (43.44%) to the peer 1, whereas the other peers received about 8% of its upload bandwidth.

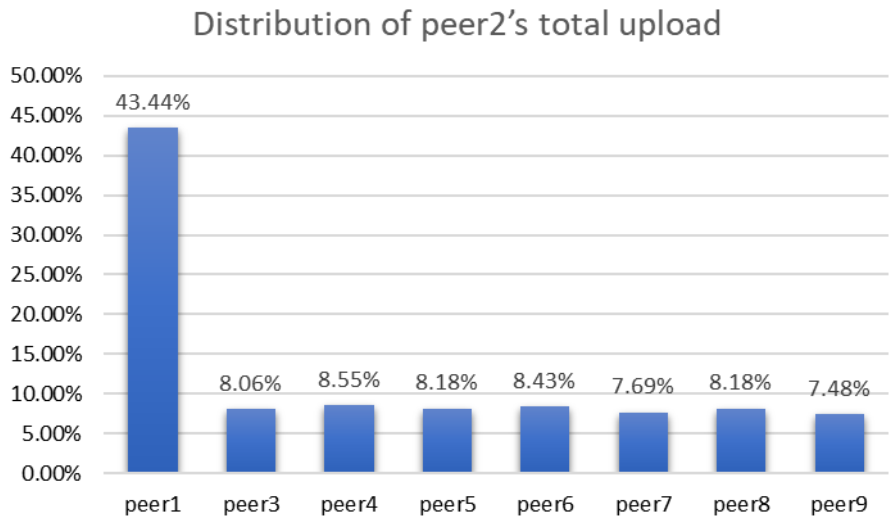

Figure 4.10 Distribution of Peer 2's total upload

We also acquired the upload bandwidth distribution of the seeder. Figure 4.11 indicates that seeder threated all the peers equally and the improvement of peer 1's download speed was not introduced by seeder, and the CPU utilization of all the peers, as shown in Figure 4.12 suggested that our algorithm implementation would not consume CPU recourse at a significant level.

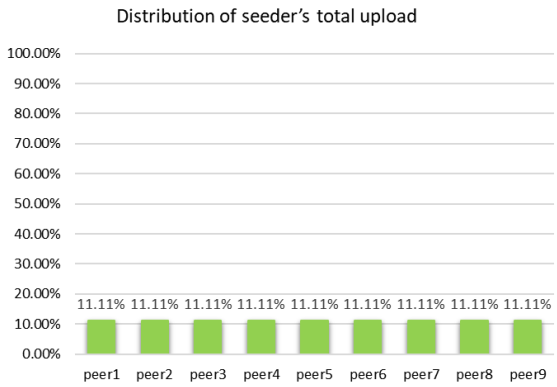

Figure 4.11 Distribution of seeder's total upload

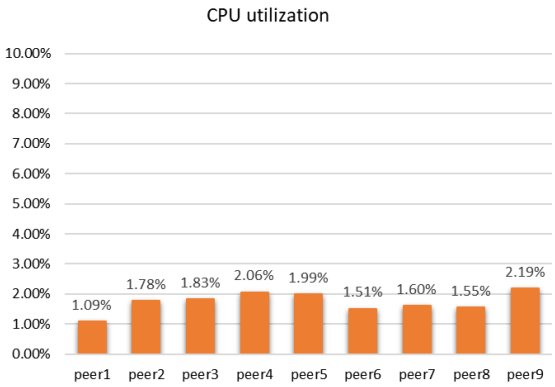

Figure 4.12 Peers' CPU utilization

4.2.2 9-peer test: 1 assisting peer with 2000 KB/s upload bandwidth

Follow the same routine as previous test, we increased peer 2's upload bandwidth to 2000 KB/S.

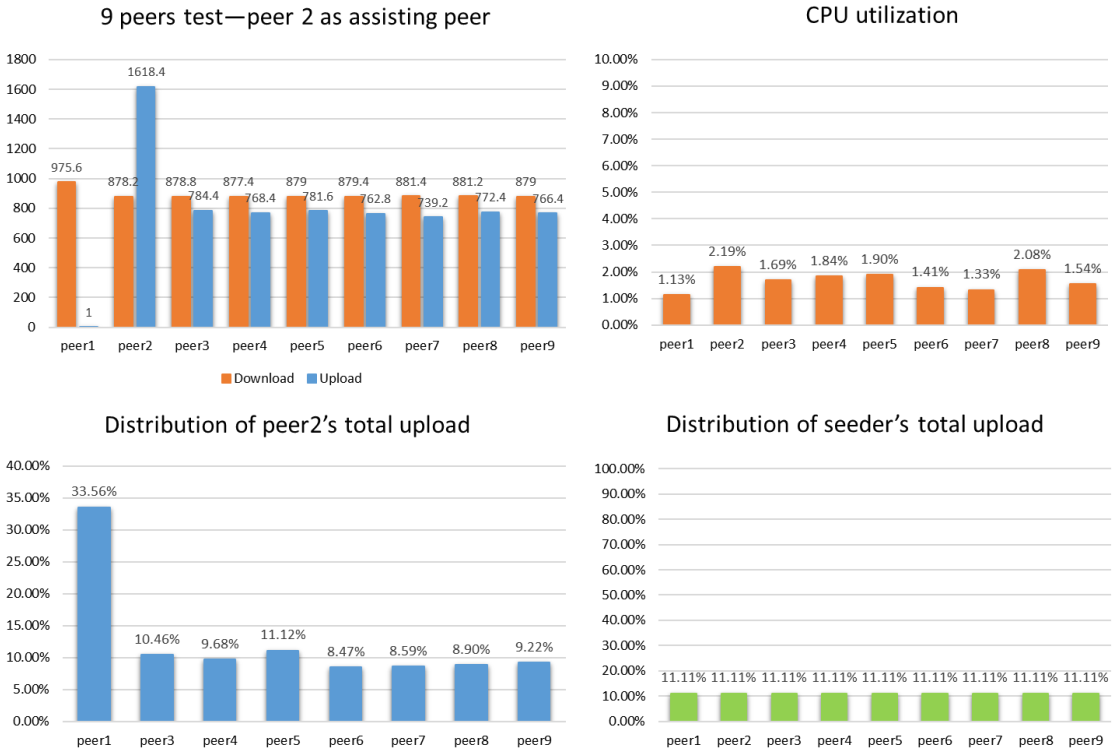

Figure 4.13, 9 peer test peer 2 as assisting peer with 2000 KB/S upload bandwidth.

	Without assisting peer	With 1 assisting peer 2000 KB/S upload bandwidth	Improvement over baseline speed	Compare against reference speed
Peer 1's average download speed (KB/S)	677.60	975.6	43.98%	99.39%

Figure 4.14 Peer 1's download speed with 1 assisting peer with 2000 KB/S upload bandwidth

Comparing to previous test, where the assisting-peer, peer 2, had an upload bandwidth of 1000 KB/S. Increasing the upload bandwidth of assisting-peer to 2000 KB/S, slow peer's uploading bandwidth further increased to 99.39% of its reference speed. Again, seeder still distributed its bandwidth equally among peers (11.11% of its total upload to each peer). However, with increased bandwidth, peer 2 decreased the portion of its bandwidth that dedicated to peer 1. This means in order to further increase slow peer's download speed, the amount of effort of assisting peer has to increase even more such that it can trade more of its own data with other peers for the data which can, later, be uploaded to peer 1.

4.2.3 9-peer test: 2 assisting peers, each has 1000 KB/s upload bandwidth

Similarly, we repeated our 9-peer test, but with 2 assisting peers this time. Both assisting peers, peer 2 and peer 3, had 1000 KB/s upload bandwidth.

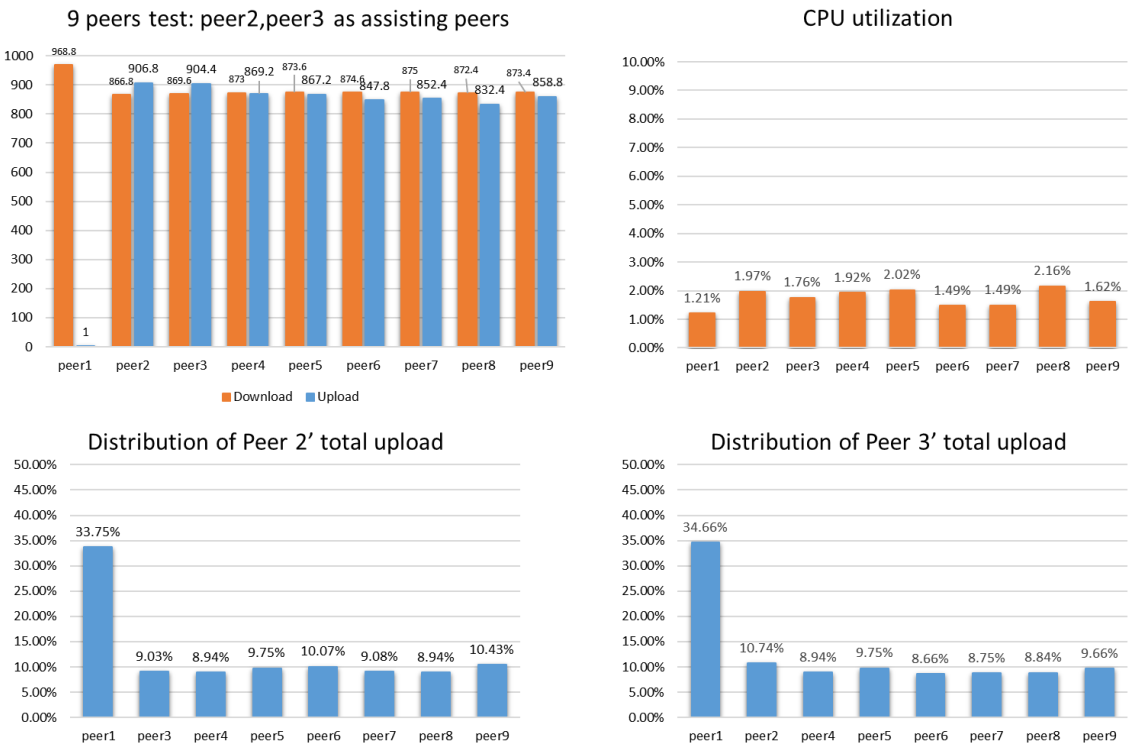

Figure 4.15, 9 peer test peer 2 and peer 3 as assisting peer with 1000 KB/S upload bandwidth

	Without assisting peer	With 2 assisting peers 1000 KB/S upload bandwidth	Improvement over baseline speed	Compare against reference speed
Peer 1's average download speed (KB/S)	677.60	968.8	42.78%	98.70%

Figure 4.16 Peer 1's download speed with 2 assisting peers with 1000 KB/S upload bandwidth

As the results indicated. Peer 2 and peer 3 contributed 33.75% and 34.66% of their total upload to peer 1. Comparing to the single assisting peer scenario, in which peer 2 had the same upload bandwidth but had a higher 43.44% contribution to peer 1. This means that in current implementation, cooperation among assisting peers is not optimal. This conclusion

of lack of cooperation can also be observed by comparing the average upload speed of peer 2 in *1 assisting peers with 1000 KB/s upload bandwidth* test where peer 2 had an average upload speed of 970.8 KB/S, whereas in this test, peer 2 and peer 3 had an average upload speed of 906.8 KB/S and 904.4 KB/S respectively.

4.2.4 9-peer test: 2 assisting peers, each has 2000 KB/s upload bandwidth

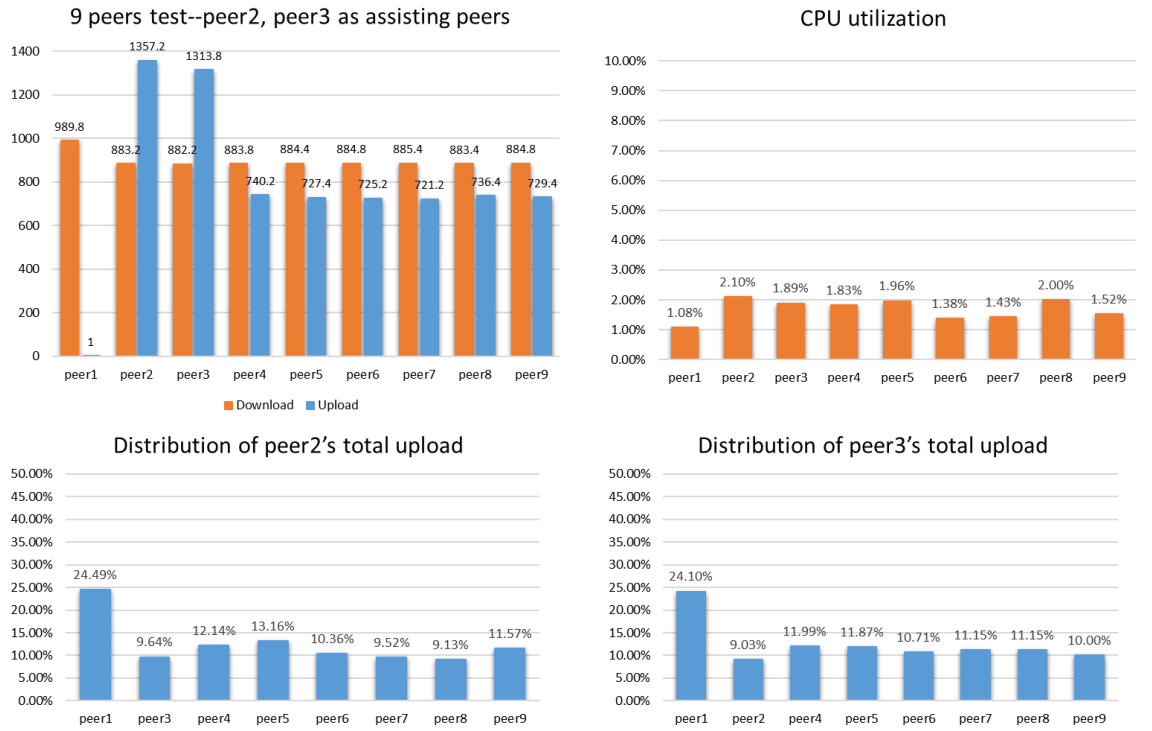

Figure 4.17 9 peer test peer 2 and peer 3 as assisting peer with 2000 KB/S upload bandwidth

	Without assisting peer	With 2 assisting peers 1000 KB/S upload bandwidth	Improvement over baseline speed	Compare against reference speed
Peer 1's average download speed (KB/S)	677.60	989.8	46.07%	100.01%

Figure 4.18 Peer 1's download speed with 2 assisting peers with 2000 KB/S upload bandwidth

As the results suggested, brutally doubling the upload bandwidth of the assisting peers can further improve slow peer's downloading speed to 100.01% of its reference speed. Clearly, our modified choking/unchoking algorithm worked as expected, and non-optimal cooperation among assisting peers can be compensated by increase upload bandwidth. However, the average upload speed of two assisting peers is still noticeably less than that

of one assisting peer working alone and this is an indication of less optimized cooperation between peer 2 and peer 3.

4.2.5 9-peer test summary

Putting the results of all previous tests together, as shown in Figure 4.19. We can comfortably answer those three questions that raised at the beginning of this section:

Given current test setup,

1) Through a try-and-error approach, we found that when the swamp size is relative small (10 to 15 peers) 1 to 2 assisting peers would be sufficient to improve the download experience of "slow" peer.

2) In current implementation, increase assisting peer's upload bandwidth would benefit the "slow" peer more than increase number of assisting peers.

3) This modified choking/unchoking algorithm can effectively redirect assisting peers' upload bandwidth to the "slow" peer. In our non-exhaustive tests, we obverse data sent to the "slow" peer can be 2 to 8 times more than the data sent to the other peers.

Peer 1 with 1KB/S upload bandwidth	With 1 assisting peer 1000 KB/S upload bandwidth	With 2 assisting peers 1000 KB/S upload bandwidth	With 1 assisting peer 2000 KB/S upload bandwidth	With 2 assisting peers 1000 KB/S upload bandwidth
Peer 1's average download speed	940.60 KB/S	968.8 KB/S	975.6 KB/S	989.8 KB/S
Improvement over baseline speed	38.81%	42.78%	43.98%	46.07%
Compare against reference speed	95.82%	98.70%	99.39%	100.01%

Figure 4.19 9-peer test summary

4.3 11-peer and 14-peer tests summary

This section shows the results of 11-peer test and 14-peer test from Figure 4.20 to Figure 4.21. we first tested with 1 assisting peer with 1000 KB/S upload bandwidth, then 2 assisting peers with 2000 KB/S.

Figure 4.20 11-peer test summary

Figure 4.21 14-peer test summary

	11-peer test		14 peer test	
Peer 1 with 1KB/S upload bandwidth	With 1 assisting peer 1000 KB/S upload bandwidth	With 2 assisting peers 2000 KB/S upload bandwidth	With 1 assisting peer 1000 KB/S upload bandwidth	With 2 assisting peers 2000 KB/S upload bandwidth
Peer 1's average download speed	889.4 KB/S	982.2 KB/S	802.8 KB/S	973.2 KB/S
Improvement over baseline speed	104.46%	125.79%	145.21%	197.25%
Compare against reference speed	91.60%	101.15%	83.69%	101.46%

Figure 4.22 Peer 1's download speed improvement in 11-peer and 14-peer test

As summarized in Figure 4.22, adding one assisting peer to help the "slow" peer could dramatically improve its download speed and this improvement become more prominent as "slow" peer gets slower with increased swamp size. Assisting peers effectively redirected their upload bandwidth to the "slow" peer as expected, but like our 9-peer test results, the efficiency of bandwidth redirection drops when utilizing more assisting peers.

In our 9-peer tests, our "slow" peer had an average download speed of 677.6KB/S download speed. Given its 1 KB/S upload speed, this baseline speed is relatively high, therefore, add 1 peer to help this "slow" peer didn't show a dramatic improvement (38.81%). However, in our 11-peer and 14-peers. Adding 1 assisting peer introduced 104.46% and 145.21% improvement.

This significant improvement difference indicates when swarm size is small and multiple peers join the swam at the same time, optimistic unchoking dominates peers' peer-discovering process. When swarm size increases "slow" peer's chance to taking advantage of other peers optimistic unchoking drops significantly, then the "unselfish" assisting' help is much more noticeably appreciated by "slow" peer.

5 Conclusion and Future Work

Integrating Cloud Computing technology into the broadly adopted BitTorrent P2P file distribution protocol enhance the performance of BitTorrent-like content distribution. This thesis work achieved its intended goal – improving the download performance of upload bandwidth limited peer in a BitTorrent-like network. The improvement we observed is quite significant. With two assisting peers in the network, we can double selected peer's download speed. The upload data distribution graphs demonstrate that our modified choking/unchoking algorithm effectively redirect peers' upload bandwidth to selected peer as well.

In real-world/less controlled environment, where availability of data among peers varies, established connections among existing peers would leave much less chance to slow peer to be optimistically unchoked, as slow peer doesn't have any data to share when it joins the swarm. Therefore, improvement on upload bandwidth limited peer brought by our proposed system and algorithm in real-world scenario may be different in a positive way.

On the other hand, improvement can be made to facilitate the cooperation among assisting peers. In this proposed system, assisting peers are not aware of others' existence, so the data exchange among assisting peers is still aimed at getting the complete content as fast as possible, which could result in unnecessary bandwidth usage between assisting peers. If communication among assisting peers achieved, assisting peers could collaborate in downloading unique data, which in return can be uploaded to the assisted peer.

Practically, this modified choking/unchoking algorithm, which is capable of letting peer who has nearly no upload bandwidth to reach its full download potential, is quite impressive, but tweaking and improvement are needed to improve its efficiency. At a system level, this thesis work does not focus on obtaining the optimal number of assisting peers in different scenarios. Finding optimal number and settings of assisting peers can be a good direction of further exploring the limit of system proposed in this work.

References

[1] Bram Cohen “Incentives Build Robustness in BitTorrent” (2003)

[2] M. Meulpolder, J. A. Pouwelse, D. H. J. Epema and H. J. Sips, "Modeling and analysis of bandwidth-inhomogeneous swarms in BitTorrent," 2009 IEEE Ninth International Conference on Peer-to-Peer Computing, Seattle, WA, 2009, pp. 232-241.

[3] BitTorrent Protocol URL: http://www.bittorrent.org/beps/bep_0003.html

[4] T. H. Luan, X. Shen and D. H. K. Tsang, "BitTorrent under a microscope: Towards static QoS provision in dynamic peer-to-peer networks," 2010 IEEE 18th International Workshop on Quality of Service (IWQoS), Beijing, 2010, pp. 1-9.

[5] I. Foster, Y. Zhao, I. Raicu and S. Lu, "Cloud Computing and Grid Computing 360-Degree Compared," 2008 Grid Computing Environments Workshop, Austin, TX, 2008, pp. 1-10.

[6] BitTorent URL: https://www.bittorrent.com/company/about

[7] D. Serain, "Client/server: Why? What? How?," International Seminar on Client/Server Computing. Seminar Proceedings (Digest No. 1995/184), La Hulpe, Belgium, 1995, pp. 1/1-111 vol.1.

[8] H. Wang, J. Liu, and K. Xu. ``Exploring BitTorrent peer distribution via hybrid PlanetLab-Internet measurement". In: Quality of Service (IWQoS), 2010 18th International Workshop on. IEEE. 2010.

[9] D. Li, Z. Chen, H. Liu and A. V. Vasilakos, "Maya-Pyramid: A Scalable and Self-Organizing Unstructured P2P Overlay," 2007 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology - Workshops, Silicon Valley, CA, 2007, pp. 411-414.\

[10] S. Rieche, K. Wehrle, O. Landsiedel, S. Gotz and L. Petrak, "Reliability of data in structured peer-to-peer systems," 2004 International Workshop on Hot Topics in Peer-to-Peer Systems, Volendam, Netherlands, 2004, pp. 108-113.

[11] Lei Guo1, Songqing Chen2, Zhen Xiao3, Enhua Tan1, Xiaoning Ding1, and Xiaodong Zhang1 “Measurements, Analysis, and Modeling of BitTorrent-like Systems”

[12] Arnaud Legout, Nikitas Liogkas, Eddie Kohler, Lixia Zhang “Clustering and Sharing Incentives in BitTorrent Systems”

[13] Michael Piatek∗ Tomas Isdal∗ Thomas Anderson∗ Arvind Krishnamurthy∗ Arun Venkataramani† “Do incentives build robustness in BitTorrent?”

[14] Kyung Wook Hwang∗ , Vishal Misra† , and Dan Rubenstein “Stored Media Streaming in BitTorrent-like P2P Networks”

[15] C. Lo and Y. Su, "P2P video streaming replication scheme for P2P VoD services," 2014 International Conference on Information and Communication Technology Convergence (ICTC), Busan, 2014, pp. 391-393.

[16] Amita Ajith Kamath, Chirag Jamadagni, K. Chandrasekaran “VirtTorrent: BitTorrent for Inter-VM File Distribution” ICC '16 Proceedings of the International Conference on Internet of things and Cloud Computing, Article No. 20

[17] Peter Mell , Tim Grance “The NIST Definition of Cloud Computing” National Institute of Standards and Technology Sept,2011

[18] R. Buyya, C. S. Yeo and S. Venugopal, "Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities," 2008 10th IEEE International Conference on High Performance Computing and Communications, Dalian, 2008, pp. 5-13.

[19] R. d. R. Righi, V. F. Rodrigues, C. A. da Costa, G. Galante, L. C. E. de Bona and T. Ferreto, "AutoElastic: Automatic Resource Elasticity for High Performance Applications in the Cloud," in IEEE Transactions on Cloud Computing, vol. 4, no. 1, pp. 6-19, 1 Jan.-March 2016.

[20] K. R. Jackson et al., "Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud," 2010 IEEE Second International Conference on Cloud Computing Technology and Science, Indianapolis, IN, 2010, pp. 159-168.

[21] M. Li, J. Yu and J. Wu, "Free-Riding on BitTorrent-Like Peer-to-Peer File Sharing Systems: Modeling Analysis and Improvement," in IEEE Transactions on Parallel and Distributed Systems, vol. 19, no. 7, pp. 954-966, July 2008.

[22] Speedtest Market Report 2018 URL: http://www.speedtest.net/reports/united-states/ , Speedtest Market Report 2017 URL: http://www.speedtest.net/reports/united-states/2017/#fixed