IMPROVING TRANSMISSION EFFICIENCY AND SCALABILITY FOR PEER-TO-PEER LIVE STREAMING

by

ZEMENG LI

Submitted in partial fulfillment of the requirements

For the degree of Master of Science

Thesis Advisor: Dr. Shudong Jin

Department of Electrical Engineering and Computer Science

Case Western Reserve University

August, 2010

CASE WESTERN RESERVE UNIVERSITY

SCHOOL OF GRADUATE STUDIES

We hereby approve the thesis/dissertation of

Zemeng Li

candidate for the Master of Science degree *.

(signed) Prof. Shudong Jin
(chair of the committee)

Prof. Michael Rabinovich

Prof. Swarup Bhunia

(date) May 24th, 2010

*We also certify that written approval has been obtained for any proprietary material contained therein.

Contents

List of Tables

List of Figures

Improving Transmission Efficiency and Scalability for Peer-to-Peer Live Streaming

Abstract

By

ZEMENG LI

Live streaming applications, especially those based on peer-to-peer networks, are becoming popular nowadays. It is widely known that there are still some performance challenges on transmission and scalability in peer-to-peer live streaming system. This thesis focuses on improving transmission efficiency in live media streaming and improving scalability in peer-to-peer live streaming systems.

First, we improve transmission efficiency in live media streaming by studying chunk scheduling algorithms which include Greedy, Rarest First, Mixed, Random and our proposed Alternate algorithms, and delivery methods which include Push and Pull methods. Based on the evaluation of startup latency and streaming continuity for different chunk scheduling algorithms and delivery methods, we discuss how to make an optimal choice for better transmission efficiency. Second, we improve the scalability for peer-to-peer live streaming system by utilizing our incentive model, a bank incentive model, which can encourage peers to make more contribution in order to obtain extra benefits from their neighbors and the system. As well as applying encouragement to the peers, our incentive model can support multiple platforms and the extensibility of incentive strategies.

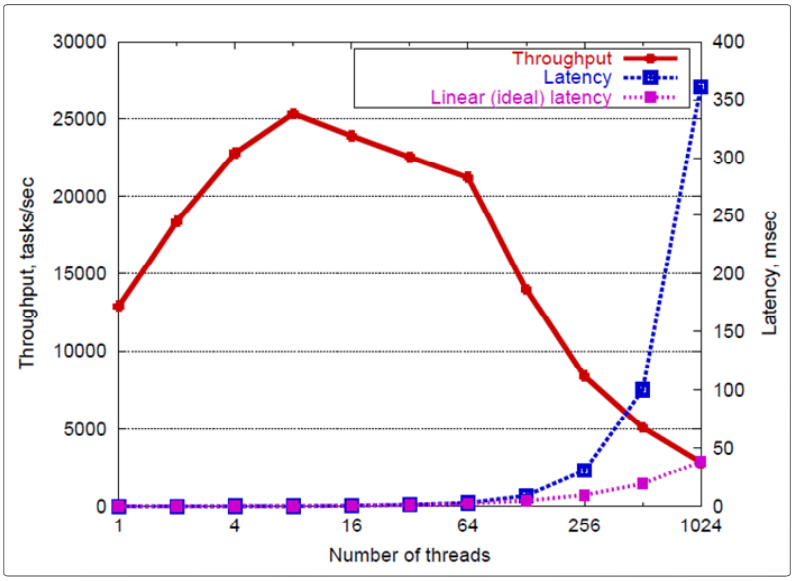
Chapter 1

Introduction

1.1 Motivation

Nowadays, an increasing number of companies produce various live media streaming applications, for example, Youtube, Helix server at RealNetworks [15], Darwin server at Apple [18], PPstream [4] and PPLive [3], along with the trend that live media streaming applications are widely accepted by Internet users.

The author participated in the development of a live streaming application at RealNetworks, This project, named Media Delivery Service (MDS) inspired by SEDA [8], is a staged event-driven architecture, which is intended to support concurrent user demands and simplify the construction of well-conditioned services. SEDA architecture makes a good contribution to controlling mechanisms and load conditioning, but the weakness of this architecture, resulting in scalability and capacity problem, makes its performance as poor as other centralized architecture in supporting a large number of users. Furthermore, users today have much higher requirements for live streaming applications. Figure 1.1 [8] shows a sharp increment of latency when the number of simultaneous users is increasing in SEDA system. We observe when the number of concurrent requests is increasing, the latency is increasing sharply, and service queue is also becoming much longer.



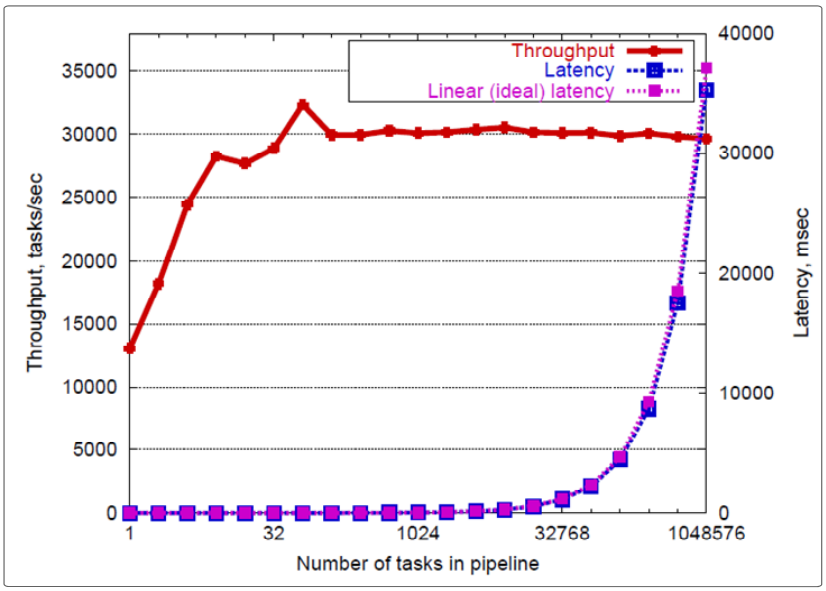


Figure 1.1 throughput degradation and performance of tasks pipeline in SEDA [8]

To solve the scalability problem, a better solution in terms of a decentralized and unstructured structure is utilized instead of the centralized one. Peer-to-Peer (P2P) networks are naturally a good option. In a P2P network, the capacity of supporting a large

number of users comes from the uploading bandwidth of all peers. Meanwhile the P2P system is able to guarantee an acceptable playback quality in most peers.

In our work, one of our goals is to improve the chunk transmission efficiency in P2P live streaming system. The second goal is to design a good incentive model which can be utilized to encourage peers to make contribution. In this way, we could improve the scalability of supporting a large number of users by the achievement of above two goals.

1.2 Idea and Contributions

To achieve our goals, firstly we evaluated chunk scheduling algorithms which include Greedy, Rarest First, Random, Mixed, and our proposed Alternate algorithm. Chunk scheduling algorithm is the core of the data-driven protocols [17], [13], [12], [20], [21]. We simulated these algorithms in C++, conducted experiments in different scenarios, and analyzed their characteristics. Meanwhile, we studied delivery methods which include Push method and Pull method. We conducted simulations for delivery methods in C++ by integrating chunk scheduling algorithms. By comparing different scheduling algorithms and delivery methods, we got the direction of improving transmission efficiency, which can lead us to choose optimal chunk scheduling algorithm and delivery method based on the design requirements for our P2P live streaming applications.

Second, besides the improvement of transmission efficiency, we propose an incentive solution, a bank incentive model for P2P live streaming system. Our bank incentive model establishes a reputation system on top of the live streaming applications by integrating itself with online activities of the peers, which can monitor a peer's

downloading and uploading, encourage them to make contributions, and help them get more benefits from their neighbors and the system. Furthermore, our incentive model can be used for non-system-performance issues and support multiple platforms. It is particularly useful for a commercial live streaming application.

1.3 An Overview of the Thesis

The remainder of the thesis is organized as follows. In chapter 2, we provide an introduction to P2P live streaming and P2P incentives. Chapter 3 describes the research on chunk scheduling algorithms and delivery methods. In chapter 4, an incentive model, a bank incentive model for P2P live streaming is described, and we demonstrate a comprehensive evaluation for this model. Finally, chapter 5 summarizes the thesis.

Chapter 2

Background and Related Work

2.1 Peer-to-Peer Live Streaming

There are many researches on the engineering of live media streaming from a server to clients along with the wide usage of Internet live streaming in our daily life. The development of encoding and decoding techniques makes many live streaming applications capable of supporting high-quality media streaming. However, serving a large number of users simultaneously challenges the scalability of P2P live media streaming.

In order to improve scalability, a solution based on IP multicast was proposed [22] in early 1980's. IP multicast substituted repeatable packets-sending at server by utilizing the routers in the network to manage the distribution and replication of media content from one source to multiple receivers. Popular IP Multicast routing protocols include DVMRP, MOSPF, CBT, PIM-SM etc. Obviously IP Multicast is an efficient solution since all data transmission is performed only once at all links. However, it has many weaknesses [25], [26], [27]. For example, IP multicast is based on IP layer, and it needs the support from network architecture. IP multicast has security and management problems because it is open to any multicasting source. It has difficulties for reliability control and congestion control, because IP multicast provides best-effort service.

Application Layer Multicast [26], [10] is regarded as a better solution which was proposed in the early of 1990's. Application layer multicast gains many advantages from

its ability that it multicasts data at the application layer by computing, replicating and transmitting at the application-layer end point. Researchers summarized Application Layer Multicast solutions as three general catalogs: Tree-based, Mesh-based and implicit [10], [26], [27], [28]. However, no matter which catalog we utilize, Application layer multicast has limited ability to support a large number of users, and it does not solve the scalability problem very well [11], [30].

Recently, with the rapid development of P2P techniques, live media streaming based on P2P networks becomes a hot topic. As another better solution, P2P technique demonstrates its capacity to solve the scalability problem of live media streaming by supporting a large number of users and high simultaneous demand. Some commercial application are widely accepted by users, for example PPLive and PPStream, and they demonstrate that P2P techniques are another better solution to solve the scalability problem of live media streaming, which is supposed to support a large number of users and high simultaneous demands. The first widely used P2P system is Napster [31], [32], and the early research was mainly focused on the file sharing and distributed hash tables (DHT). DHT is used to construct a special structure for all peers, and hash tables are distributed into each node. Chord, CAN, Pastry and Tapestry belongs to this area. For the P2P file sharing, most applications are widely used nowadays, for example, Gnutella, BitTorrent, KaZaA, eMule.

P2P file sharing applications are growing rapidly, which inspires researchers to focus on supporting live media streaming based on the P2P networks [33], [34], [35]. It also inspires researchers to apply similar idea in P2P file sharing that media content is divided into chunks for transmissions based on sharing protocols, for example BitTorrent. P2P

networks for live media streaming becomes a hot topic because of its capability to solve the scalability problem of supporting a large number of users.

When P2P techniques for live media streaming were well developed, many scheduling and delivery algorithms were proposed at the same time, for example data-Driven algorithms, mesh-based algorithms, swarming-based algorithms, and pull-based algorithms. These protocols use random process to pick neighbors for a peer when it constructs overlay. For the streaming, they use the similar idea like BitTorrent: media content is divided into many chunks. Each peer periodically sends content information to its neighbors. When peers receive the notifications from their neighbors, they explicitly request chunks from their neighbors who may already store those chunks. In each peer, a buffer window is used to manage the chunk requests, uploading and downloading. In this way, a peer is a requester who requests media content for playing back, as well as a provider who provides content to its neighbors.

2.2 Peer-to-Peer Incentives

P2P live streaming applications have many advantages compared against traditional media streaming, and more users choose to join this community. Most of these applications are developed as an unstructured overlay network based on data-driven protocols. Since P2P system is an open system and users are free to join and leave, researchers are inspired to design incentive mechanisms to solve management problems and improve scalability. For example, in eDonkey and eMule, each peer maintains a metric for every other peer, proportional to the size of the file it obtained from them. The metric determines serving priorities of other peers in that peer's queue [36], [37]. KaZaa

utilizes a participation level metric which is defined as the ratio of uploaded to downloaded file sizes and determines the file search radius [36], [38]. One implementation of incentive mechanism is through a reputation mechanism, which attempts to estimate the behavioral profile of a peer in its transactions with other peers by recording and studying a peer's online activities. In [39], peers use direct and second-hand information to assess the behavior of other peers. With a statistical approach, erroneous or incompatible reputation rankings are prevented from inclusion in reputation updates [36]. In [40], the reputation of a peer is based on its past interactions and on indirect information from a selected subset of peers through a weighted voting scheme and can be used for bandwidth allocation or determination of query time-to-live (TTL) duration [36], [41]. In [42], an iterative distributed algorithm for trust information aggregation is provided, where each peer computes the maximum positive eigenvalue of the trust matrix. The eigenvector corresponding to that eigenvalue is the global trust vector of peers. Peers use trust values to select servers and avoid downloading inauthentic files [36]. In the model of [43], each peer is characterized by its reputation that models its past transaction behavior, as well as by its inherent tendency to cooperate. The reputation of a peer converges to its true inclination to cooperate [36]. Finally, [44] shows that certain server selection and contention resolution policies need to be adopted by peers when acting as clients and servers respectively in order for a reputation scheme to fulfill its purpose [36].

In order to improve transmission efficiency and scalability, we are not only focused on the optimization of scheduling and delivery algorithms, but there are also some challenges we need to deal with in P2P incentives. We simply suppose all peers can

obtain an optimal playback quality as long as the total upload capacity supply is higher than the system's minimum bandwidth demand.

For example, there are 10000 simultaneous users in the system. If a user watches a video with 360p resolution, it requires 700kbps download bandwidth to fluently play this video. The total bandwidth demand to support all 10000 users is 7 Gbps. 7 Gbps is almost a huge number for a server. However, 360p is just the lowest video quality. With the rapid development of H.264 codec, video quality has already gotten to 1080p from 480p/720p. We can define

System Efficiency Ratio = Total Upload Capacity / (#users * min Bandwidth Demand)

The more upload bandwidth a system has, the more user and better playback quality it can support. We observe that a critical challenge may hinder the system's efficiency. No matter how we improve transmission algorithms, the system needs more upload capacity to improve scalability and to support a large number of users. Inspired by this observation, we propose that a good incentive strategy for the P2P live media streaming is extremely important, which should be able to encourage peers to make more uploading during online status.

Chapter 3

Improving Transmission Efficiency in P2P Live Streaming

3.1 Problem Description and Motivation

P2P live streaming utilizes similar idea as BitTorrent, which is the most important work in the P2P community. In both of them, a media is divided into chunks for transmission. However, P2P live streaming has much difference from P2P file sharing. On one hand, P2P live streaming demands highly due to its real-time requirements. For example, P2P live streaming requires low startup delay, smooth playback, and good video quality. On the other hand, P2P live streaming does not require a complete copy of the media when user is ready to play back a video. Even when only a small part of content is ready, user can start watching the movie immediately.

The reason for the difference between P2P live streaming and P2P file sharing lies in demanding and transmission algorithms. A user requests the most demanded chunks to satisfy the real-time demand in P2P live streaming. These most-demanded chunks are managed and maintained by buffering at each peer. In other words, the chunks in the buffer are those that are ready to be played immediately. Streamed media is divided and packetized into chunks for transmission when the media is requested. The user's media player manages a buffer window for downloading and feeding the playback process. Users can play not only a continuous media time range, but they can also forward or backward or jump-to a time point. Corresponding to user's request, media player adjusts its demanded media range. This results in updating the buffer window. Generally there are two basic steps for streaming a media. In the first step, a peer selects the demanded

chunks by chunk selection algorithms. When the media player confirms the chunks it will download, it explicitly requests these chunks from its neighbors. In the second step, neighbors forward the demanded chunks to the requester when they have available copies. During this step delivery method takes the major responsibility.

The scheduling algorithms and delivery method are extremely critical to a live streaming system, and they are also the core of data-driven approaches. Furthermore, streaming quality, such as smoothness, delay, video quality and transmission speed depends on scheduling algorithms and delivery methods.

The necessity that researchers need to improve playback continuity and streaming quality, reduce delay, and improve delivery efficiency is enhanced by the trend that users raise their requirement for live media streaming, and meanwhile live media streaming applications are becoming more bandwidth consuming. It is helpful that we clearly discuss the impact of scheduling algorithms and delivery method to the P2P live streaming systems.

In this chapter, we study different chunk scheduling algorithms, chunk delivery methods, and their potential effect on the system. Then we propose a new chunk selection algorithm. By clarifying their difference based on the evaluations, we are able to make a better choice and get optimal efficiency when we design a P2P live streaming system.

3.2 Chunk Scheduling Algorithms

In this section, we present the ideas of chunk scheduling algorithms. Every peer maintains a buffer window *W* that can cache up to *n* chunks. We give a definition that *W(i)* is the i_{th} chunk in the buffer window, where i = [1, n], i.e. W(1) is the left-most chunk in the buffer, and W(n) is the right-most chunk in the buffer. The window *W* is always moving forward along the media content. In other words, the buffer acts as a sliding window which covers the demanded media range. Each buffer position is initially empty, and will be filled by the chunk scheduling algorithms. When the empty positions in the buffer are filled, the peer is able to playback the media content in the buffer window. In the P2P networks, all peers form a community. Every peer is a requester who requests and downloads the chunks from its neighbors. Meanwhile, it is also a provider who is willing to upload its available chunks to other requesters. As a result, chunk scheduling can be generally modeled as a pull process, in which each peer selects another peer as a neighbor to download chunks which are not available in its local buffer window.

Chunk scheduling algorithm determines the order of the demanded chunks from W(1) to W(n). Simply it means that chunk scheduling algorithm determines which chunk will be downloaded first, and which chunks will be downloaded next. There are several popular chunk scheduling algorithms.

Greedy algorithm aims to fill the empty chunks in the buffer closest to the playback deadline in the first order. It means that a peer will request and download the chunks which are closest to playback deadline in the top priority.

From a single peer's point of view, greedy algorithm is intuitively the best algorithm for streaming. Since peers are focusing on the short-term playback needs, it has a significant advantage in minimizing the startup latency. However, from a system wide view, especially when the peer population is large, greedy algorithm has its disadvantages. For example in improving the playback continuity, greedy algorithm is worse than rarest first algorithm. To be described below, here we present the pseudo-code of greedy:

```
Greedy (W)
FOR i = 1 to n    // n is the buffer size
      IF W(i) is empty    // W(i) is the i^th position of buffer window
          Select W(i);
          Return;
```

Table 3.2.1 Greedy Algorithm

Rarest First algorithm selects a chunk which has the minimum number of holders among the neighbors in the first order. The rarest first algorithm works as follows. Each peer maintains a list of the number of copies of each chunk in its peer set. It uses this information to pick out the rarest chunks. Let *m* be the number of copies of the rarest chunk, then the index of each chunk with *m* copies in the peer set is added to the rarest chunk set. The rarest chunk set of a peer is updated periodically. Each time a peer selects the next chunk for downloading randomly in the rarest chunk set [16]. Rarest First algorithm ensures high diversity and availability of the chunks, such that it is effective in improving system-wide playback performance. The following is the pseudo-code of Rarest First algorithm.

```
RarestFirst (W)

FOR i = n to 1    // n is the buffer size
      IF W(i) is rarest    // W(i) is the i^th position of buffer window
         S ← W(i) ;  // S is the rarest chunk set
random-select (S)
Return;
```

Table 3.2.2 Rarest First Algorithm

Mixed algorithm [9] is a combination of rarest first and greedy algorithms. In mixed algorithm, the buffer window W will be partitioned by a demarcation point m. Initially the Rarest First policy will be used in first-order on the right part of buffer which is divided by the demarcation point. If no chunk can be downloaded using rarest first policy, the Greedy is used on the left part of buffer in the second order. Mixed algorithm can take advantage of both rarest first and greedy. By devoting a fraction of the buffer, it can achieve better continuity and lower startup latency.

```
Mixed (W)
FOR i = n to m    // m is the mixed point
      IF W(i) is rarest    // W(i) is the i^th position of buffer window
         S ← W(i) ;
IF S is not empty
     Return random-select(s);

FOR i = 1 to m − 1
      IF W(i) is empty
         Select W(i) ;
         Return;
```

Table 3.2.3 Mixed Algorithm

Random algorithm, as indicated by the name, randomly selects the index of empty chunks in the buffer. We present the pseudo-code of random algorithm.

***Random* (W)**

While(true)

$i = random(n)$;

IF $W(i)$ is empty // $W(i)$ is the ith position of Buffer Window

Select $W(i)$

Return

End While

Table 3.2.4 Random Algorithm

So far we discuss 4 different chunk selection algorithms. There are two basic aspects they are dealing with, one is to reduce startup latency, and the other one is to improve playback continuity. Rarest First algorithm is much better in dealing with scale, and greedy is better in playback performance in small scale networks. Both of them have their advantage and focuses. The intuition of their different advantage inspires us to propose a new chunk selection algorithm: Alternate algorithm. In alternate algorithm, we utilize greedy policy and rarest first policy in an alternate order. Firstly, alternate algorithm selects the next empty-chunk which is closest to the playback deadline. In the next time, it selects the rarest chunk for downloading. Secondly, alternate algorithm goes back to the first step and starts the loop again. The idea of alternate algorithm is to use buffer space to improve startup latency and playback continuity performance simultaneously. The following is the pseudo-code of alternate algorithm.

***Alternate* (W)**

Initiation
$i = 1$; // for Greedy
$j = n$; // for Rarest First
AlgType = Greedy;
Iteration:
While(true)
IF AlgType == greedy
IF $W(i)$ is empty // $W(i)$ is the ith position of buffer window
Select $W(i)$;
$i = i + 1$;
AlgType = Rarest First;
Return;
ELSE $i = i + 1$;
ELSE IF $W(j)$ is rarest
Select $W(j)$;
$j = j - 1$;
AlgType = Greedy;

Table 3.2.5 Alternate Algorithm

3.3 Evaluation of Chunk Scheduling Algorithms

Users often rates a system based on some basic qualities: playback continuity, delays, transmission speed and media quality. Since users care about transmission speed mostly by evaluating buffering time, startup delay and continuous playback, here we use two major metrics to evaluate the performance of a P2P streaming application:

Startup Latency: startup latency refers to the delay time a user should wait before starting playback.

start-up latency = (the time of 1st playback)- (start time)

Continuity: continuity means the degree of interrupt (delay) during playback. We define it as the percentage of delay time in total playback time:

*playback continuity = (1-(total delay time/finish time)) *100%*

Note: the total delay time does not count in the first chunk's delay.

To study the performance and impact of a variety of chunk scheduling algorithms, we implemented these algorithms in the simulator which we programmed in C++. To simplify the design, in our simulator the network topology is generated by a uniform distribution model. That means the neighbors of a peer is uniformly assigned for that peer initially.

The following is the design of our simulator. In the simulation, there is a group of hosts, which represent peers. Each peer maintains a list of available and demanded chunks. Server serves as the Tracker in the P2P system. Metrics are used to collect information from the server and all peers. TestParameter is used to conduct simulation experiments.

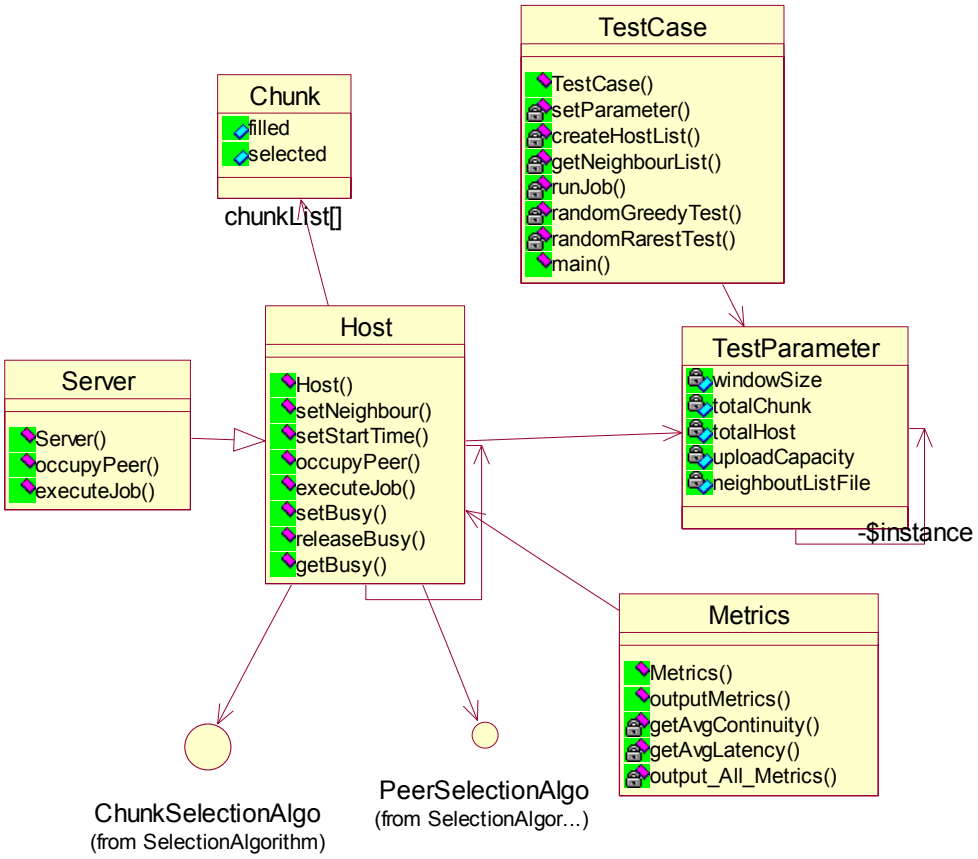


Table 3.3.1 Class relationship in simulator design

Table 3.3.2 shows the testing configuration in our simulations.

Parameter	value
Peers	100
Chunks	1000
Demarcation	Mid-point of buffer
Buffer size	20 ~ 100

Table 3.3.2 simulation configuration for evaluating scheduling algorithms

First, we start our simulation with different peer's population. In this simulation, two scenarios are used to study the efficiency of chunk scheduling algorithms and impact on overall network performance on different network size. With the same simulation

configuration, one scenario is used to study the startup latency in different chunk scheduling algorithms. The other scenario is used to study the continuity. In each scenario, the buffer size of every peer is the same, which is equal to 20. The unit of buffer size is not actual data size, but a relative one. Here, it means that the buffer is able to contain 20 chunks in total. The sliding buffer window only slides when all chunks in the window are filled. To guarantee the fairness, every peer requests and playbacks the same media file, which has 1000 chunks of size in our simulation, and all peers start downloading at the same time. To simulate a distributed system using a single thread program, the running order of peers is randomly shuffled at each time slot.

Figure 3.3.1 shows the average startup latency of chunk scheduling algorithms for the experiments with different peer population in the networks. In Figure 3.3.1, we observe that Greedy algorithm's startup latency is very low since it is focused on satisfying the immediate playback demand.

Theoretically, Alternate algorithm deals with most-demand chunks and rarest chunks in alternate order, so it handles the startup latency in half of its effort. As we observe in the figure, Alternate algorithm's startup latency is also very low, and its performance is almost the same as the Greedy algorithm.

Rarest First algorithm has the worst performance on startup latency. Because Rarest First is not focused on the most-demanded chunks, but on the rarest chunks in the networks with the first order, so it is better in improving the overall performance of the networks and average continuity at each peer. In the Figure 3.3.1 and Figure 3.3.2 we can observe such expectation that Rarest First's startup latency is worst when it is compared with the

other four chunk scheduling algorithms. Moreover, the startup latency increases with the increasing number of peers. The reason is that all peers start up at the same time slot and only server has the complete media source at the beginning, so more peers in networks will induce severer competition. As a result, the total time of successfully filling the whole buffer window will be much longer.

In the Mixed algorithm, we set its demarcation point at the middle of buffer window in our simulation. It also shares the same network configurations which are used in the simulations for Greedy and Rarest First. We observe in Figure 3.3.1 that the average startup latency with mixed algorithm is nearly half of Rarest First algorithm, and the trend of curve is also similar. Since Mixed algorithm is a pure combination of Greedy and Rarest First algorithms, Mixed algorithm's startup latency is between Greedy and Rarest First.

Compare with the other four chunk scheduling algorithms, the average startup latency with Random algorithm is not stable, but it is better than Rarest First and Mixed in dealing with the startup latency. However, when we evaluate the Random algorithm from the aspect of continuity, Random algorithm is also a good algorithm.

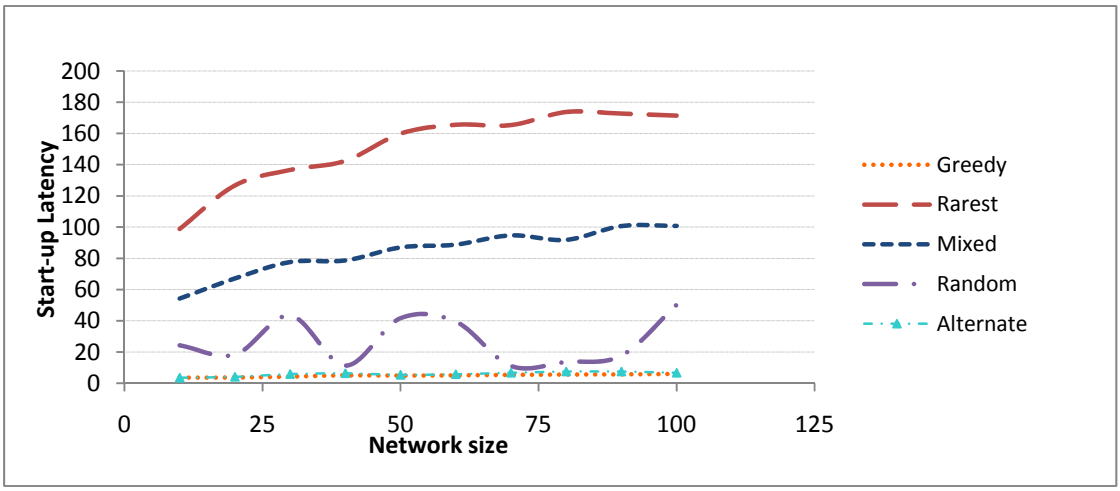


Figure 3.3.1 Average startup latency comparison in different network size

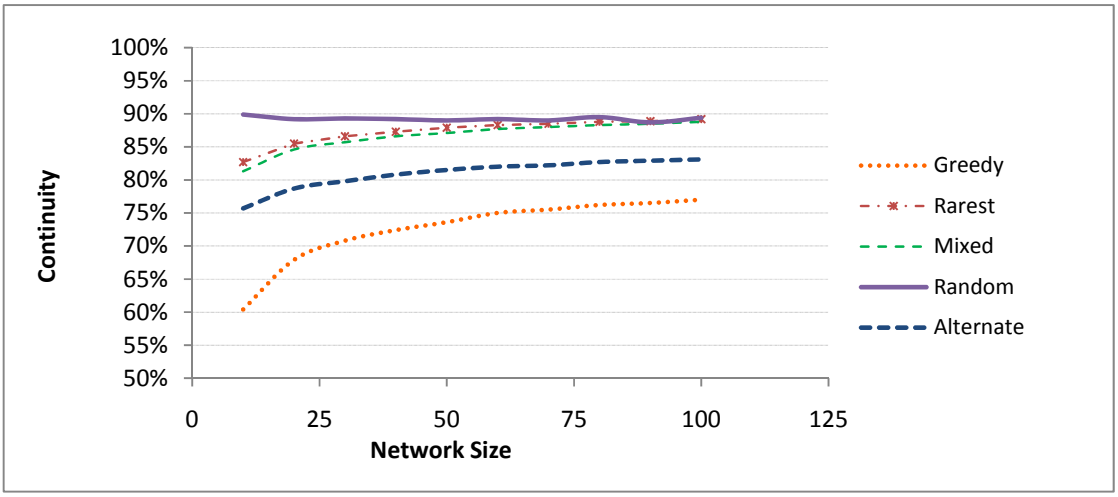


Figure 3.3.2 Average continuity comparison in different network size

In the second scenario, we study the performance and efficiency on continuity of different chunk scheduling algorithms. In this scenario, the simulation configuration is the same as we test in the first scenario. Buffer window size is 20 chunks, and the media source size is 1000, and every peer requests, downloads and playbacks the same media source. In Figure 3.3.2 we observe that, for most chunk scheduling algorithms, except Random algorithm, the quality of continuity slowly becomes better with the increasing

number of peers. As we mentioned before, all peer start up at the same time and use same chunk scheduling algorithm. More peers in networks will increase the diversity of chunks and improve the availability of chunks. As a result the general continuity will increase.

It is very interesting that Random algorithm's performance is very good, especially for small-size networks. It seems there is contradiction that, if we evaluate continuity, Random algorithm is better than Rarest First algorithm which should be theoretically better than Random. It is probably due to our simulation configuration. Since in our simulations the networks size is not very large, in each time slot if the chunks a peer select fail to be filled, the peer will try next chunk until success. Therefore the continuity of Random algorithm is possible to overcome other algorithms, because Random is able to increase the diversity of chunks. However, if we set the network size to be large enough, i.e., there are more diversity and plenty of chunks, the performance of Rarest First is better than Random algorithm.

For Greedy algorithm, its performance is the worst, because Greedy always focuses on the most-demanded chunks, not the continuity. However, with the increasing number of peers, its performance increases more rapidly than the other chunk scheduling algorithms. The competition among peers will not become stronger and it is more likely that each peer can successfully download the chunk it requests in Greedy.

Alternate algorithm's performance is better than Greedy algorithm, because it tries to download the most-demanded chunk for once in every two time slot. It is very interesting that Rarest First algorithm and Mixed algorithm's continuity are very similar. Mixed algorithm does not improve continuity as well as we expect.

Based on our experiments, generally we can conclude that Rarest First algorithm and Mixed algorithm have no advantages on startup latency in either large or small networks. Alternate algorithm and greedy algorithm are best evidently.

Second, we study the performance of chunk scheduling algorithms in the conditions of changing buffer window size. In these simulations, we also test these algorithms in two scenarios as we use in testing different network size. One scenario is focused on the startup latency, and the other scenario is focused on continuity. Since in this set of simulations we are focused on evaluating the performance with different buffer size, so the network's configuration is the same, that peer's population is the same in all simulations. Furthermore, the media source keeps the same in order to guarantee the fairness, whose size is always 1000 chunks. The sliding buffer window only slides when all empty positions are filled. All peers start their jobs at the same time. To ensure the fairness, the running order of peers is still randomly shuffled at each time slot.

Figure 3.3.3 shows the average startup latency of different chunk scheduling algorithms with changing buffer size. Since chunk scheduling algorithm is used to fill empty positions in the buffer window, by changing the buffer we evaluate the efficiency and performance of these chunk scheduling algorithms. In Figure 3.3.3 we observe that Greedy algorithm is always the best algorithm in handling startup latency, since its first priority is to download the most-demanded chunks in the buffer. We also observe that Alternate algorithm is almost as good as Greedy algorithm. In this Figure we can see that Rarest First algorithm has the worst performance on startup latency, as we expect Rarest First algorithm is not good at reducing startup latency. Since Mixed algorithm is a combination of Greedy and Rarest First, its performance is between them. For Rarest

First algorithm and Mixed algorithm the latency increases with the increasing buffer size linearly. The number of chunks which are downloaded before the user starts the first playback depends on the buffer size in Rarest First algorithm and in Mixed algorithm, and the startup latency for the first played chunk also depends on buffer window size. If the buffer size is bigger, then the startup latency could be longer. Compared with Rarest First and Mixed algorithms, the startup latency of Greedy/Alternate/Random algorithms is independent of the buffer window size.

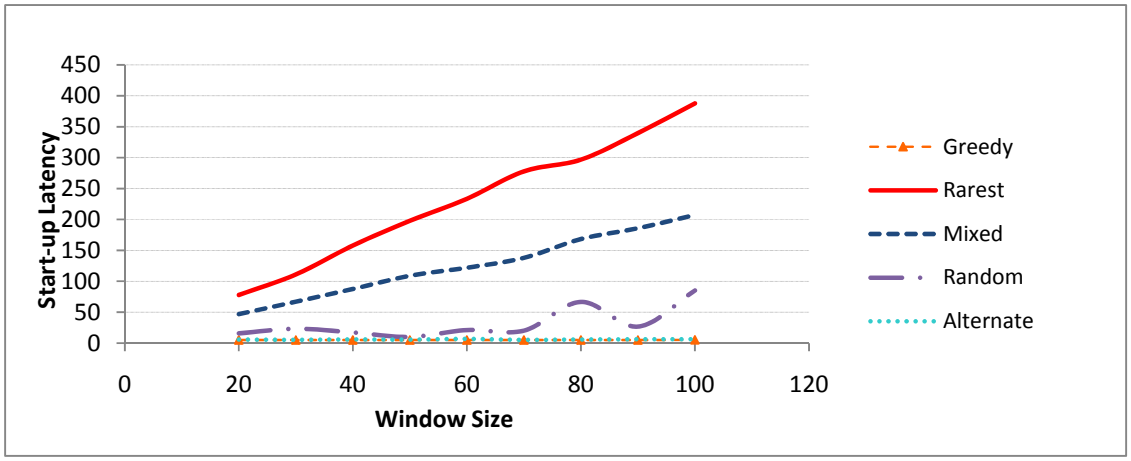


Figure 3.3.3 Average startup latency comparison in different window size

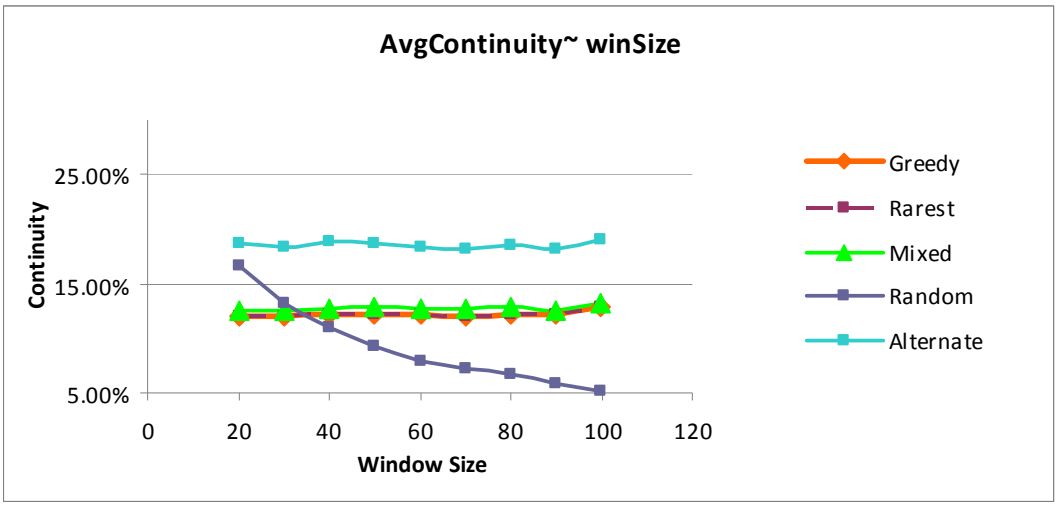


Figure 3.3.4 Average continuity comparison in different window size

The experiment result of continuity is similar as we study in evaluating startup latency. For most of the algorithms except for Random, their continuity keeps constant when the window size is increasing. It means the continuity is independent of the window size. Considering the evaluations on changing network size, we notice that network size has a significant impact on continuity. Because if there are more peers in the networks, they increase the diversity of chunks, but the buffer window size is not relative with the continuity.

All of above testing cases observe the metrics when jobs of all peers are finished. Now we study the metrics during the playback. In this simulation, the population of peers is set to be 100, media source size is 1000 chunks, and we set buffer window size to be 40. Since startup latency only determined by the time slot of first playback, so here we only evaluate the continuity curve at different time slot in Figure 3.3.5. In this figure, we find that all algorithms' trends are similar. The continuity increases with time, and the continuity become stable after a particular time. Therefore, we discover that when the number of chunks achieves a particular number, the continuity will stay on a stable and good value.

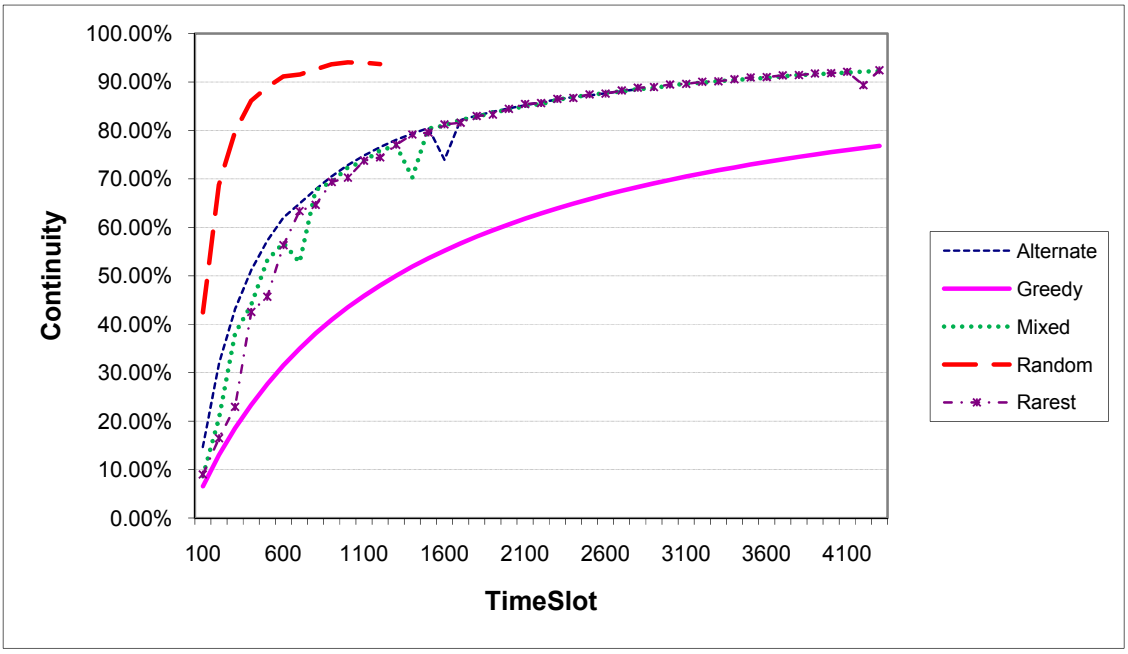


Figure 3.3.5 Continuity during playback

3.4 Chunk Delivery Methods: Push versus Pull

In order to improve the transmission efficiency in P2P live streaming system, in this section we study two delivery methods: Push method and Pull method. As we discuss in section 3.3, chunk scheduling algorithms are typically data-driven methods, which are also pull-based protocols. In this type of protocols, media content is divided into chunks for transmission. Each peer periodically exchanges content-map notifications with its neighbors. Meanwhile, this peer explicitly requests empty chunks from its neighbors based on the information in the content map. Obviously, the major drawback of pull method is the control overhead and delay. Each peer needs to notify its neighbors of the chunks it has. Based on the notifications its neighbors receive, they are able to download

demanded chunks. These communication messages incur some control overhead. Meanwhile, it becomes the source of delay.

A good solution for these problems is to integrate the push method with the pull method. A brief description about Push-Pull delivery method is as follows:

- First, each peer uses the pull method as a startup,
- Second, after a startup when chunks are received, a peer will receive the next chunks from its neighbors without explicit demand.
- Third, when a peer receives a chunk, it will relay/push this chunk to those neighbors who might be interested with this chunk.

To improve transmission efficiency and reduce delay, Push method has significant advantages compared again Pull method: since a peer is no longer to send a request for every chunk in the Push method, the provider positively sends continuous chunks to the requester based on requester's previous demand information, instead of waiting for arrivals of every demand. In this way the delay can be reduced and the diversity of chunks are also improved. Furthermore, by reducing the number of explicit requests between peers, the system reduces the control overhead, and push method efficiently utilizes the remaining bandwidth. Hence, the transmission delay will be reduced and playback continuity can be improved. Meanwhile, push method ensures high diversity and availability of the chunks. It means more copies of media content are distributed into peers.

3.5 Evaluation of Delivery Methods

To study the performance of Push and Pull delivery methods, we use playback continuity we define in section 3.3 as our major metric. Since Push method utilizes the Pull method as an initial step, Push method is the same as Pull method in reducing startup latency. As a result, we do not consider the startup latency in this scenario. In our experiment, there is one tracker, peers and one log-server used for collecting data for analysis.

Initially, tracker is supposed to connect with all peers, and each peer has several neighbors (20% of population and randomly picked), and sources (a complete copy of a file) are distributed randomly to some peers. Although at an initial step each peer has randomly-picked neighbors, in our simulation each experiment for testing different strategies share the same overlay. This guarantees the correctness of comparisons. In the following set of test cases, the peer's population increases, the fluency is evaluated in pull and pull-push mode. Following figures are the experiments results.

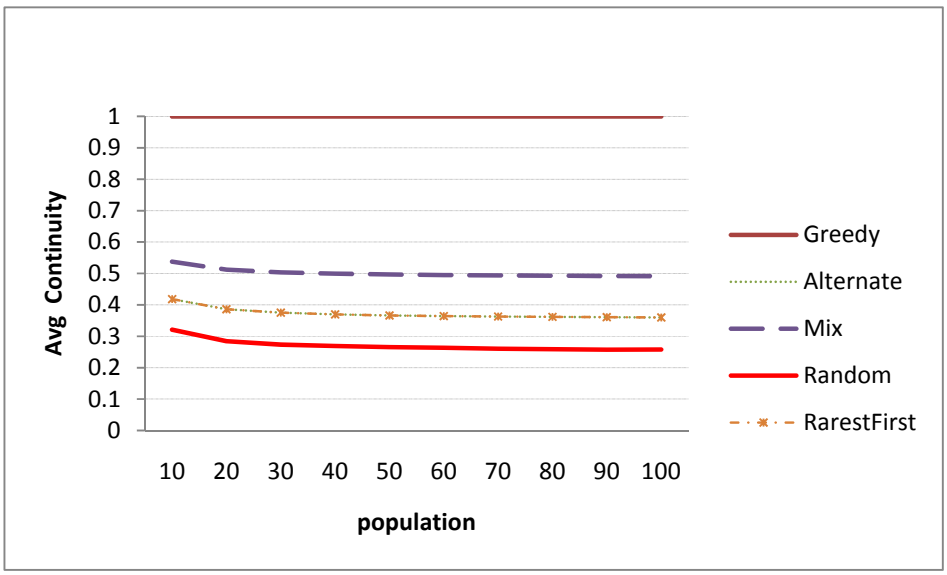


Figure 3.5.1 Average continuity in Pull method

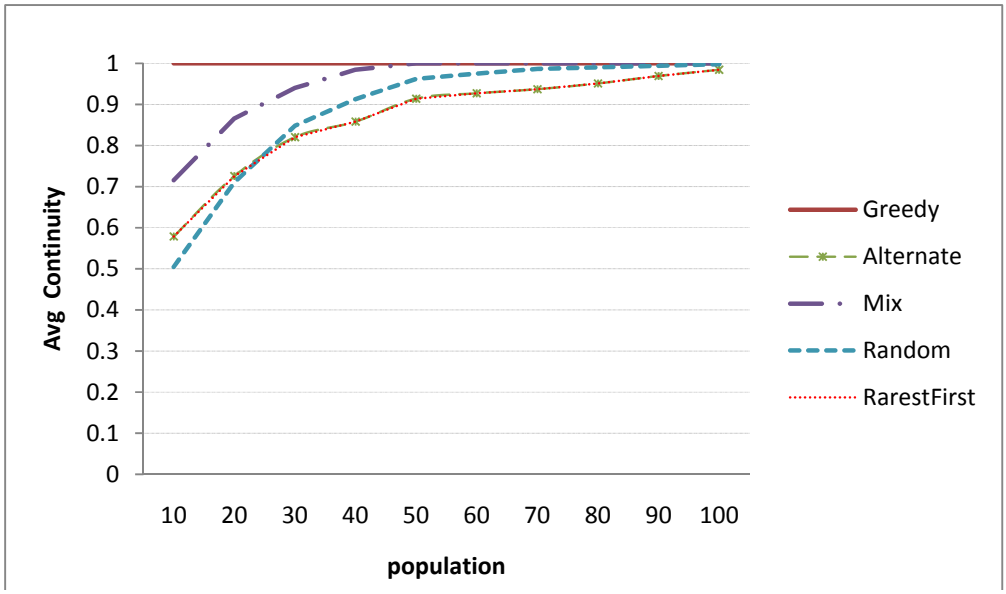


Figure 3.5.2 Average continuity in Push method

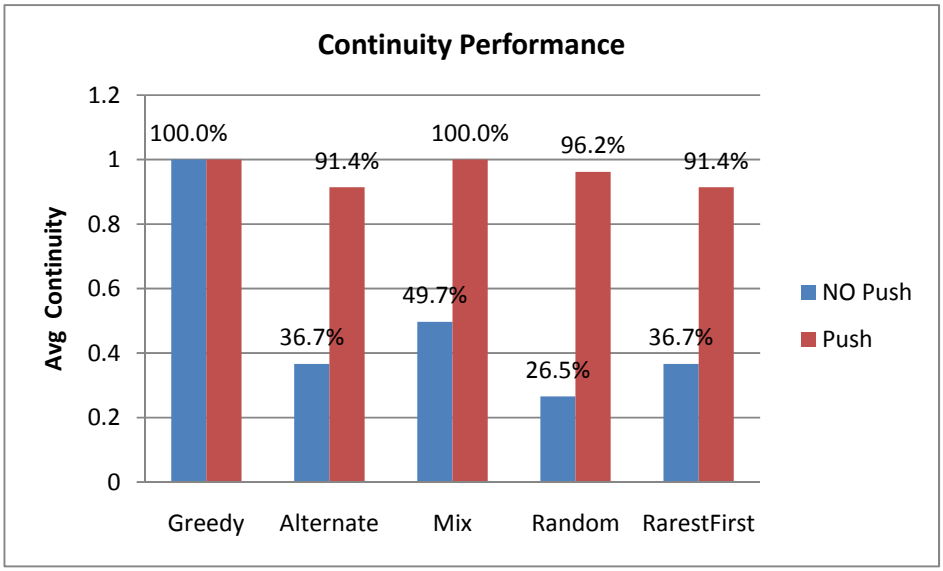


Figure 3.5.3 Comparison of Pull and Push methods

From Figure 3.5.1 we observe that, with the increase of peer's population, the playback continuity stays almost constant. It means the efficiency and performance of transmission of chunks do not become much better in Pull mode. In Figure 3.5.2, we use Push method

instead of pure Pull method. Figure 3.5.2 shows that when the peer's population is increased, the average continuity will also increase constantly till up to almost 1 (the best playback continuity). We observe that Push method has a significant impact in improving playback continuity, since Push method helps chunks distributed more broadly and rapidly. Figure 3.5.3 shows a comparison of Push method and Pull method: Push mode can increase the playback continuity. From the point view of the whole system, system has better performance in Push mode. Take Rarest First for example (in Fig. 3.5.3), the average Fluency can be increased by 149%. This is an amazing improvement for playback.

3.6 Summary

The efficiency of chunk selection algorithm plays a significant role in the P2P live streaming system. We evaluate the performance of different chunk scheduling algorithms. We observe that Greedy has the lowest startup latency, Rarest First is much better in scalability, and Greedy is much better in a small scale networks. Mixed has better efficiency in playback continuity. We designed a new algorithm – Alternate algorithm. Compared with other chunk selection algorithm, Alternate algorithm presents a comprehensively good performance.

Meanwhile, Push method is also able to improve streaming efficiency. It reduces the latency, increases the fluency, and increases the diversity of media chunks. The simulation shows that Push method has better performance in utilizing bandwidth

capacity. It can distribute media content rapidly and broadly, that indicates system has smaller overhead and users have lower delay.

Chapter 4

Improving Scalability for P2P Live Streaming

4.1 Problem Description and Motivation

In chapter 3, we discussed scheduling algorithms and delivery methods in P2P live media streaming. Based on the evaluation and study of scheduling algorithms and delivery methods, we are able to improve the transmission efficiency of P2P live streaming. In this chapter we present an incentive model to improve the scalability for P2P live streaming system.

We use an example to illustrate the scalability problem in P2P live media streaming system. For example, there are 10000 active users in the system, and all of them are streaming videos based on 360p resolution. To support their real-time demand, we calculate that the system needs about 7 Gbps uploading throughput. If the video quality increases, the system requires higher upload throughput to support all users. According to our definition:

System Efficiency Ratio = Total Upload Capacity / (#users * min Bandwidth Demand)

Except the improvement of transmission efficiency, another challenge for the P2P systems is that the support capacity depends on the overall upload throughput in the P2P live streaming system. The larger the overall upload throughput is, the more users and higher streaming quality the system can support. To improve the scalability, we need to design an effective incentive strategy which is able to encourage peers to make more uploading, so that the system can get more upload throughput.

In P2P file sharing systems, typical incentive strategies focuses on the fairness and free-rider problems, while incentive strategies should focus on encouraging peers to make contribution and offering extra benefits to good-reputation peers in P2P live streaming system. In this chapter, we present an incentive model, a bank incentive model.

4.2 Introduction to Bank Incentive Model

The idea of reputation based on a peer's contribution and consumption is widely used as a common incentive solution in many P2P systems for incentive purposes. However, most of these works focus on fairness and free-rider problems. There is no such a work so far, which provides an incentive solution especially for P2P live streaming. Our work focuses on designing incentive solution for P2P live streaming system.

Nowadays H.264/AAC codecs which can support high resolution videos are widely used in the Internet streaming. The resolution of videos varies from lower-to 360p to up-to 1080p. High resolution videos have higher real-time requirements. It is a challenge for a P2P system to support high quality video streaming, because it needs to effectively utilize remaining bandwidth at peers. To encourage peers for more contribution and utilize remaining bandwidth at peers, we propose a banking system which can be used as an incentive solution.

In this chapter, we present the advantage and features of our incentive model. First, it improves the scalability in P2P live media streaming. Second, it deals with not only system performance issues, but also non-system issues. Third, we present how to use this

model in different platforms/systems, and how to extend incentive strategies in this model.

Here we present the components of our bank incentive model, it has 3 major components:

I Peer's Bank Account:

- Deposit Account (balance);
- Credit Account (credit-in-use / credit limit);
- Credit Level
- Factors that affects a peer's the Bank Account

II Bank Working Model:

- How to support credit query
- How to maintain peer's account

III Credit Level Rate Model;

- Credit Level Hierarchy
- Credit Rating: How to evaluate a peer's credit level

The first component is the bank account of each enrolled user in the system. Every peer is encouraged to register a user ID. Otherwise, a guest has only basic benefits and functionalities, and the system will constrain a guest in many online activities. A peer's bank account includes a deposit account, a credit account, and a credit level which is considered as a peer's reputation. The bank account is relative with a peer's downloading and uploading activities. The second component is the working model which is responsible for managing and utilizing a peer's bank information. The third component is credit level rating model which is used to manage a peer's reputation**.**

4.2.1 Peer's Bank Account

Every registered peer (user) has an associative bank account in our model, and the account composes of three parts: Deposit Account, Credit Account, and Credit Level.

A *Deposit Account* stands for a peer's available balance which highly depends on a peer's income, and the income is computed based on a peer's contribution.

Credit Account includes two parts: *credit-in-use* and *credit-limit*. Credit-in-use has the meaning of consumption, which indicates the volume of chunks a peer has downloaded from its neighbors. Credit-limit, as indicated by the name, constraints the overall credit a peer can finally use. In another words, if the credit-in-use is greater than credit-limit, this peer cannot download any more from its neighbors until it pays off its credit-in-use.

Credit Level has the meaning of reputation. If a peer has higher credit level, it can be accepted by more neighbors in the P2P community and it can get more benefits from the system. The Credit Level is the core that drives the incentive strategies.

We take an example to illustrate how a peer's bank account is working. For example, when a peer named Alice joins the system, her available balance is 3000 points in her Deposit Account. Meanwhile, her credit-in-use is 800 with the credit-limit of 1500 in Alice's credit account. It means Alice has accumulated 800 point for her downloading activities, and she can continue to download (consume) 700 point until she use off all her credit-limit of 1500.

For a peer's bank account, we further define:

Account Balance = DepositBalance + Income*(1+bonus) – CreditInUse (1-reward)

When a new-comer with both of bonus and reward being zero consumes 100MB content, the incentive model requires that the new-comer needs to contribute approximate 100MB content to its neighbors in order to keep the account balanced. If the new-comer fails in its contribution, its available account balance will decrease.

When a user regarded as a good credit peer with bonus and reward being greater than zero downloads 100MB content, it could upload less than 100MB content to balance its account. To simplify description, we take Alice for the example and we assume bonus = reward = 0. Alice's Bank Account:

Alice's Status	Credit	Deposit
Initial Status	0 / 1500	0
Hours later	450 / 1500	+600
Pay credit	0 / 1500	150
Credit limit increase	0 / 1700	150

Alice's Account Balance

= DepositBalance + Income*(1+bonus) – CreditInUse (1-reward)

= 0 + 600 – 450

= 150

Initially, Alice's credit-limit was 1500. She downloaded nothing, and her saving was zero. After she downloaded 450MB content, her credit-in-use was 450. At the same time, she also uploaded 600MB content to her neighbors (making contributions). Because of her contribution, her income was 600 during this process. Later on she used her income to pay for her credit. Finally because of her contribution, she was rewarded for 200 points of credit limit add-up. As a result, Alice played back media of 450MB, and contributed 600MB media to neighbors. Her credit line increased by 200 points, and kept 150 points deposit for future usage.

Table 4.2.1 presents a variety of factors that affect a peer's bank account.

For the income, income is computed directly based on a peer's uploading (contribution). If a peer is willing to perform as a helpmate for the system, for example it provides additional service to other peers, which is supposed to be taken care of by the system. This peer gains income for its helpmate activities. If a peer is willing to be a Push-method helpmate, it also gains income for its kindly contributions.

For credit-in-use, obviously the downloading is used to compute a peer's consumption (credit-in-use). If a peer is willing to provide debit to its neighbors, it is also considered as consumption.

For credit-limit, it is directly associated with the credit level. The higher the credit level is, the higher credit limit a peer can get to.

Income (deposit)	Credit-in-use	Credit Limit	Credit Level
Contribution (Upload)	Consumption (Download)	Historical Contribution	Historical Contribution
System Helpmate	Debit		Historical Loan
Push Helpmate			Historical System Help
			Historical Pull-Push Help

Table 4.2.1, Factors affecting a peer's bank account

4.2.2 Bank Working Model

Section 4.2.1 presents the major components of the bank incentive model. In this section, we present the corporation of the components of our bank incentive model.

There are three general steps. In the first step, peers make queries and answer queries in the system. In the second step, peers download chunks based the availability of media chunks and upload chunks based the requests of their neighbors. In the final step, peers send report, and the Bank Server updates the accounts of peers. Figure 4.2.2 displays the communication flows between peers in the banking system.

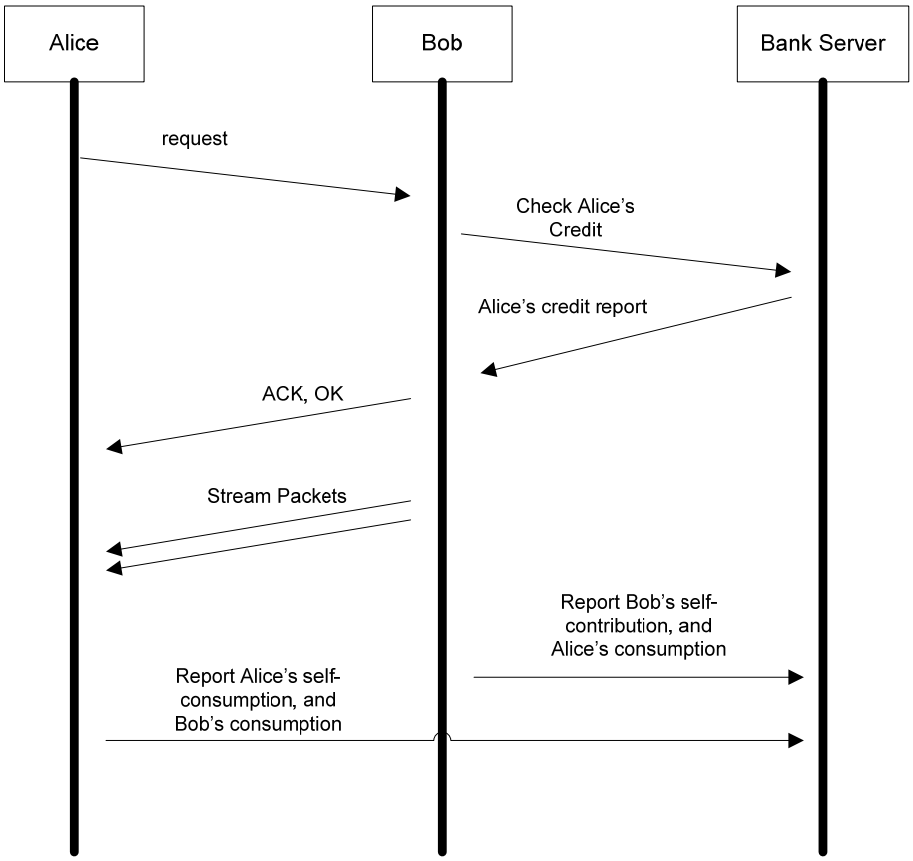


Figure 4.2.2 Communication message flow in bank working model

The following example presents how this model works out between peers. Alice wants to playback a movie, she connects to Bob as her neighbor. Then Alice sends a request to Bob, saying "I want to download some content from you". When Bob receives her request, Bob sends a credit query to Bank Server, saying "I want to check Alice's credit level". Bank Server responds to Bob the credit report of Alice. When Bob receives the credit report, he will make a decision whether or not to accept Alice's request based on Alice's credit report. If Alice's request is accepted by Bob, Bob will send media chunks to Alice. Otherwise Bob could deny Alice's request if Alice's credit level is not acceptable for Bob. Both Alice and Bob will report to the Bank Server periodically the

transaction information between them. Based on this information, the Bank Server will update Bob and Alice's bank account and credit level.

4.2.3 Credit Level Rating Model

After sorting and organizing the peers based on their credit levels in our bank incentive model, our model establishes a Credit Level Hierarchy which is the core of implementing incentive strategies in the system. With the corporation of the Credit Level Hierarchy, the bank incentive model is able to support credit query and identification by managing the credit levels in the Bank Server, which form a bottom-up hierarchy in the Bank Server. Compared with users at bottom who are identified as bad credit peers, those at top have highest credit level who are considered as good credit peers.

Figure 4.2.3 presents a general hierarchy of the credit levels of peers.

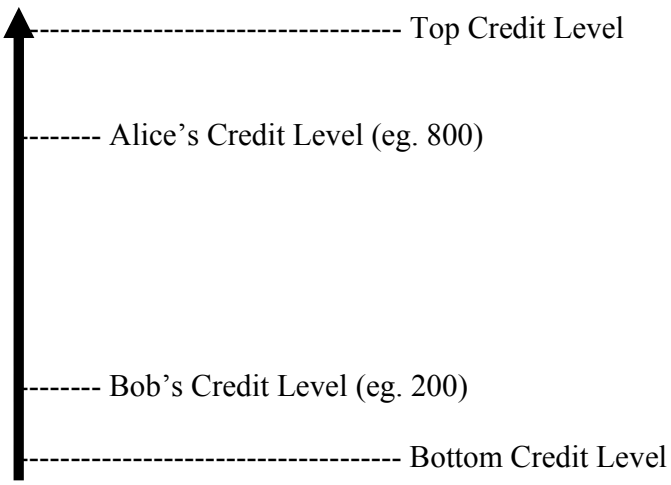


Figure 4.2.3 Credit level rate of peers

The Bank Server is responsible for organizing all peers into a hierarchy, so that this hierarchy can be used as the core of incentive strategies. For example, Alice's credit level is higher than Bob. If Bob wants to download chunks from Alice, Alice could deny his request as an option or Bob will be put behind in Alice's service queue, just because Bob's credit level is lower than Alice's. So for any peer, if it wants to obtain more benefits from its neighbors, it has to contribute. That's the reason this credit hierarchy can be used to encourage a peer's contribution as an incentive factor.
Another incentive reason is that, many bonus benefits and functions will be unlocked with the increase of Credit Level.

4.3 Improving Streaming Performance

4.3.1 Query ACK and Deny

As we present the working model at 4.2.2, a peer can deny a requester's demand if the requester has a lower credit level after the peer looks up the credit information at the Bank Server which is responsible for managing a peer's credit.

For example, Alice connects Bob as her neighbor. She sends a request to Bob in order to download chunks from Bob. When Bob receives her request, Bob will check Alice's credit level at the Bank Server. If Alice's credit level does not meet Bob's expectation (in our model, this expectation is ranged based on Bob's self-credit-level by a lower-bound threshold), Bob will deny Alice's request.

In this scenario, a peer is encouraged to improve its credit level by making contributions (uploading) in various ways. Otherwise its credit level will be far away behind its neighbors' credit level, and most of its neighbors could deny its request.

4.3.2 Priority in Service Queue

In our bank incentive model, we utilize priority policy as another incentive strategy, which is the most important incentive strategy in our model, since priority policy is also widely used even in the real world, for example vip customer in shopping or waiting room. A user with better credit level is considered as the one that has higher priority in our incentive model, and it could be served by its neighbors in first order. Normally, a peer has a limit of the maximum number of peers which it can support in a unit time, (no matter it's a software-default value or a user-customized value). Priority policy does not concern about the arriving order of the requesting messages, but concern about the credit level of the peer who sends this request message. This feature ensures that a request could be processed in a first order even it arrives in a later order. Every peer maintains a service queue which keeps the requesting messages from its neighbors.

For example, Alice is a provider, and she provides chunks to her neighbors: Bob, Charlie, and Daisy. Suppose, credit_level(Daisy) > credit_level(Bob) > credit_level(Charlie). Alice receives Bob's request and Charlie's request in 1st order and 2nd order respectively. Alice receives Daisy's request in the last order. However Daisy's credit level is the highest. Even though Daisy's request is the last-come, her request will be processed in a first order by Alice. If Alice receives the requests which exceed the maximum number

that Alice can support, the requests from those lower-credit-level peers will be suspended till next time.

In this scenario, if a peer is always keeping a lower credit level, its request message could fall behind others in its neighbor's service queue. Furthermore, if a request message is timed out in the service queue, this message will be dropped. Priority policy formulates a competition scenario, which encourages every peer to make more contribution in order to increase its credit level.

4.3.3 Group Selection

The bank incentive model, providing query ack/deny and priority policy as incentive strategies we have discussed in previous sections, also supports group selection that is another incentive strategy in our model. Group selection can block unexpected neighbors and select better neighbors for a peer. Take Alice as an example (Figure 4.3.3).

Alice has a good credit level. When Alice sends requests to the Tracker, the Tracker answers to Alice, and provides the neighbor information to Alice, or when Alice receives gossip message from her neighbors directly, typically a set of random neighbors is selected. However, in our model group selection can help Alice to select a range of neighbors which could be better than random selection. For example, this range could be $(1 \pm 20\%)$ of Alice's credit level. In this way, some unexpected neighbors for Alice will be filtered out.

As a reversed-side example, if Bob's credit level is always bad, then the kind of the neighbor information he can get is always comparatively bad group. Or we can rule that,

Bob is able to fetch all kinds of neighbor information. However, if Bob's credit is not located at a given-range of Alice's credit, Alice will deny Bob's request.

In this scenario, the banking (incentive) system encourages peers to contribute to improve their credit scores.

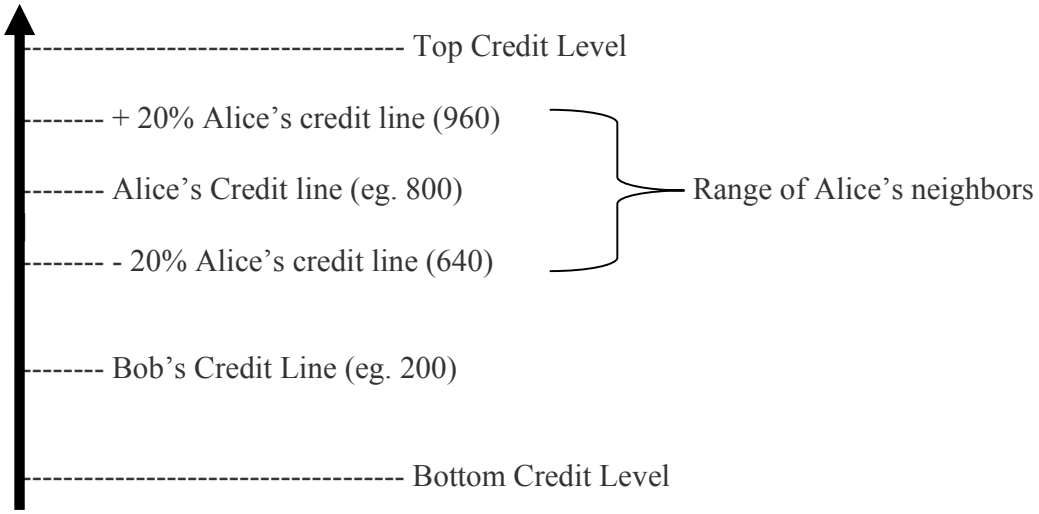


Figure 4.3.3 Group selection

4.3.4 Push Helpmate

In chapter 3, we discussed Pull delivery method to Push delivery method. Compared with Pull method, a peer using Push method is able to play a good role in helping its neighbors, which is the major advantage of Push method. In our incentive model, a peer will be awarded if it is willing to help its neighbors. To illustrate how a peer could be rewarded for its contribution as a push helpmate, we take an example. As a push-method helpmate, Alice receives packets from Evan, and then Alice will forward these packets to her

neighbors (Bob, Charlie and Daisy). However these packets are not those ones that Alice is explicitly requesting for herself. Alice is helping with pushing packets to her neighbors, and improving system performance.

In this scenario, although Alice is downloading content from Evan, she does not need to pay for this consumption. Meanwhile, Alice is uploading (forwarding) packets to neighbors, so she is able to accumulate her income.

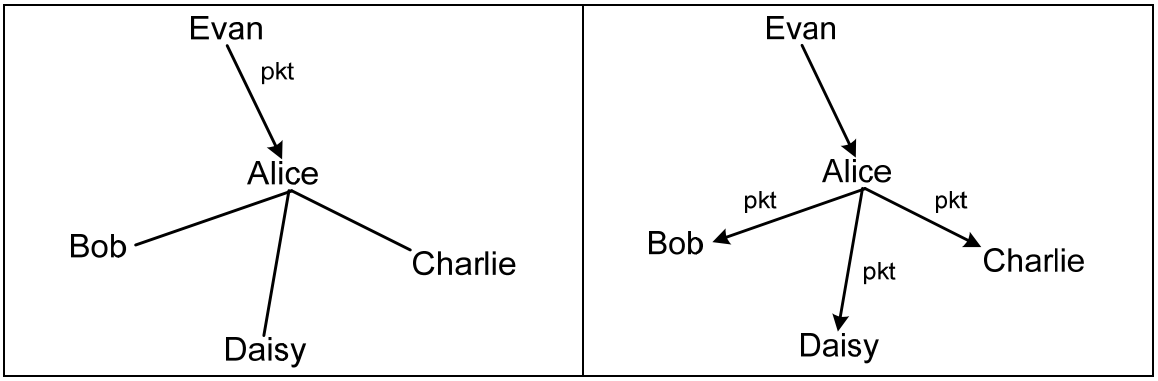


Figure 4.3.4 Push helpmate

4.3.5 Contribution Bonus and Consumption Reward

The bank incentive model is running on top of the P2P live streaming system by monitoring a peer's downloading and uploading activities, establishing a consumption and payment system, and integrating incentive strategies with itself. When a peer is downloading chunks from its neighbors, it is actually consuming its credit. In the opposite side, a peer is earning income when it is uploading chunks to its neighbors. A peer uses its income to pay off its consumption. When a peer's credit level increases

because of its contribution, it will get higher bonus which makes the peer earn more income based on the same amount of contribution, and meanwhile get higher reward which helps the peer make less payment for the same amount of consumption. Here we can image both of these incentive functionalities in the scenario of using a credit card in our daily life.

Figure 4.3.5 presents a general relationship between credit level, bonus and reward:

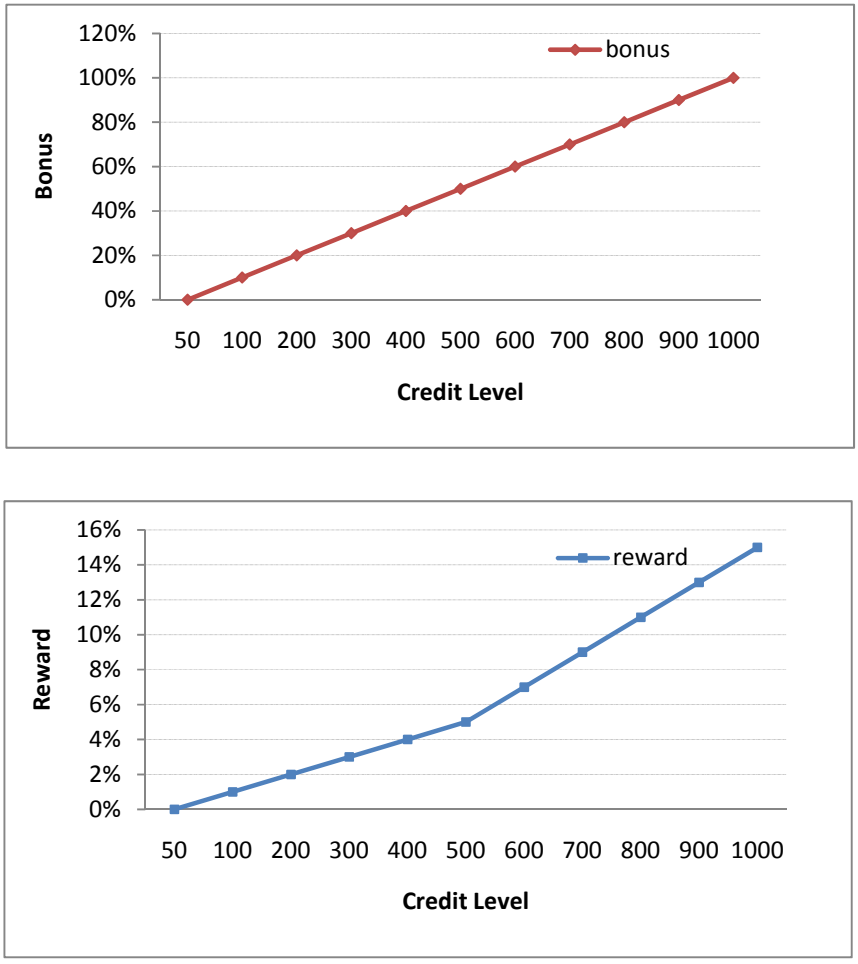


Figure 4.3.5 Changes of Bonus and Reward based on credit Level

Peers are encouraged to increase their credit level by contribution, in order to get more benefits in terms of Bonus and Reward.

4.4 Improving E-commerce Functionalities

In Section 4.3, we present typical incentive functionalities for our incentive model that can encourage peers to make more contribution, which improves the system's scalability and efficiency to support a large number of users. In this section, we present that our model can be also used as an incentive in the non-system issues, commercial functionalities.

Various user-customized functionalities are supported in many live streaming applications, for example, vip user and advertisement during playing. Based on the research of some popular streaming applications, we summarize their functionalities for non-system-performance issues. We present that our bank incentive model can be utilized for e-commerce purpose.

Streaming Applications	E-commerce Functionalities
easyMule	Advanced search Share personal file
Youtube	Ads in buffering
PPStream	Ads in buffering (non-vip) NO Ads (VIP) VIP movie
PPLive	Broadcast personal media Support third-party application Personal setting for VIP
Real's Helix	File conversion for multiple devices (not free)
Apple's Darwin	File conversion for multiple devices (not free)

Table 4.4.1 E-commerce functionalities in popular commercial streaming applications

Besides supporting incentives to improve scalability, the bank incentive model can be used in the area of E-commerce functionalities, which is another contribution for encouraging peers for wide online activities. For example, Table 4.4.1 demonstrates various functionalities in different live media streaming applications. PPStream provides registration and purchase for vip users who have special benefits and bonus. Inspired by this feature, we can build a special-benefit switch-on hierarchy based on the credit level of the peers. With the increment of credit level, special benefits and functionalities will be unlocked, which means that the higher the credit line is, the more functionalities a peer can get.

These live streaming applications categorize users by user's level or by user's payment type. Typically a user registered as a payment user or a user with higher level, gets more benefits. In our model, extra benefits are associated with different credit levels. Figure 4.4.2 presents this organization. While the credit level is increasing, more functionalities are unlocked which are supposed to be locked at lower level. If a peer is trying to increase its credit level, it obtains extra functionalities.

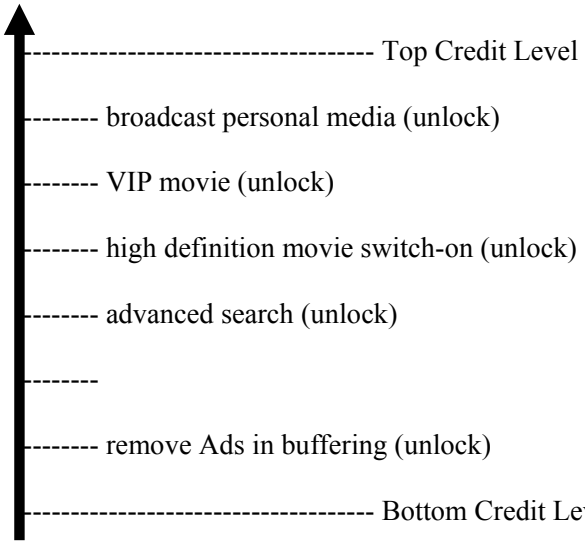


Figure 4.4.2 Unlock functionalities based on credit level

4.5 Extension of Bank Incentive Model

4.5.1 Share Credit in Different Platforms/Systems

Since the bank incentive model is a centralized model, in which every user is encouraged to register a unique ID, so its information can be kept in the server and used in a wide area of platforms or systems. For example, a peer's credit can be used in a mobile system.

If multiple machines are associated with the same user id, the user can choose one machine as the major source of its contribution, and select another one for more consumption. For example, user can use the desktop PC for major uploading activities, and choose the mobile device as the major consumption device since the bandwidth at the mobile device is more valuable.

4.5.2 Extend Incentive Strategies

Not only the bank incentive model is compatible in different platforms, but we can also extend its incentive strategies when we need to re-design or add incentive strategies. Here we mimic the idea of Strategy Design Pattern to extend the incentive strategies for our bank incentive model. Since our model built on top of P2P system inherits to be open, we can add or modify incentive strategies in the ways of affecting a peer's credit level. For example, when Alice communicates with Bob, or Alice is a system helpmate, or Alice works as a Push helpmate, changes on Alice's credit depends on the methods that are used to evaluate a peer's credit. Although the communications are fixed, we can still

modify or add the methods of How-to evaluate a peer's contribution, consumption and reputation level.

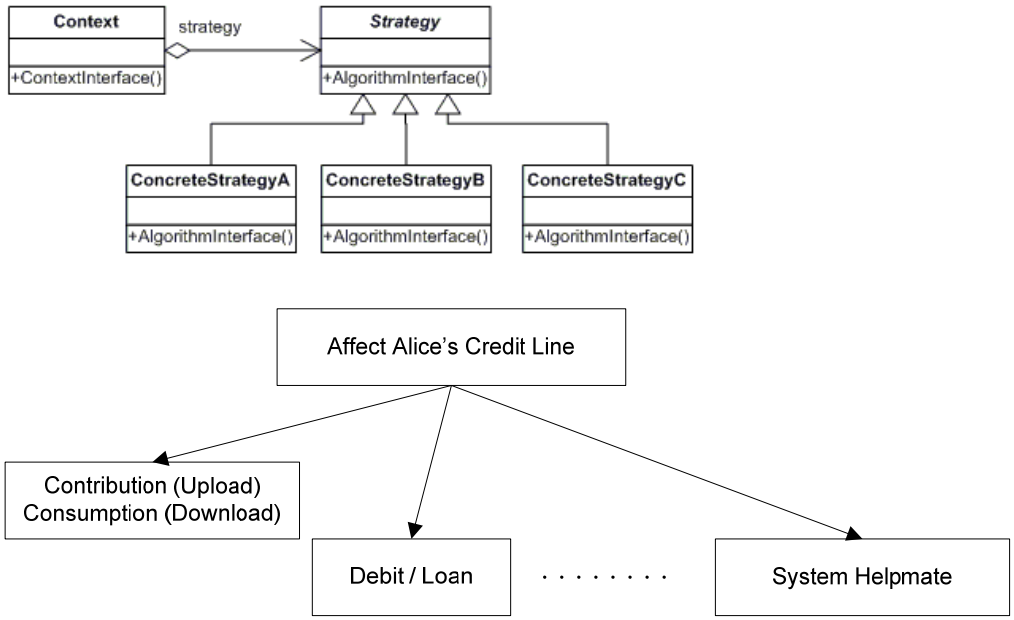


Figure 4.5.2 Extend incentive strategies in the Bank Incentive Model

4.6 Simulations and Results

To study the performance and impact of this incentive model on P2P live media streaming system, we implemented a simulator in Visual Studio C++ 2008, in which we simulated a group of peers which connect with randomly-picked neighbors of a given number, a Bank Server which is responsible for managing the banking information of every peer and answering queries from the peers, and a Track Server which provides media and peer information. Table 4.6 shows the network configuration for incentive strategy experiments in our simulations. Instead of the actual units, the unit of these parameters and settings are relative units but constant. For example, buffer size is 100. It

means the buffer window can hold 100 chunks in total. The download speed is 60. It means the speed is 60 chunks per time slot. The timing is not an actual time unit, but time slot.

Peer's Configuration	
Peers Population	30 ~ 300
Buffer Size (chunks)	100
Download Speed Range	1 ~ 100
Upload Speed Range	1 ~ 100
Media Size	10000
Running Time Slot	1 ~ 100
Peer's Account Configuration	
Credit Limit	1000
Credit Level	500
Deposit	500
Income	0
Credit-in-use	0
Bonus	0 ~ 100%
Reward	0 ~ 15%

Table 4.6 Simulation configuration for evaluating bank incentive model

4.6.1 Impact of Upload Bandwidth

In this simulation, we intend to find the impact of upload bandwidth on the download bandwidth at each peer when we utilize the incentive model in P2P live media streaming system. Theoretically we expect that, a good incentive model has a critical feature of the capability of encouraging peers to obtain better download bandwidth by making more contribution.

In our experiments, a group of peers with the same download bandwidth configuration but different upload bandwidth are deployed in the P2P system which provides a basic P2P environment. For this group of peers, we configure their download bandwidth at 50 chunks per time slot along with different upload bandwidth.

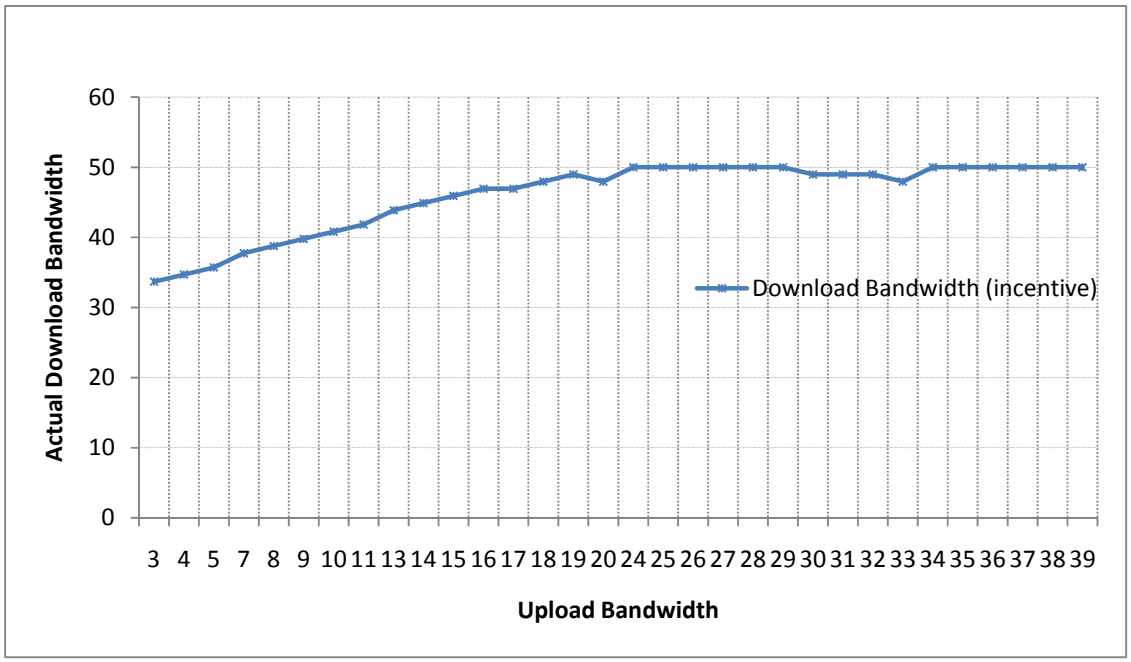


Figure 4.6.1-1, Actual download bandwidth with increasing upload bandwidth

Figure 4.6.1-1 shows the changing curve of actual average download bandwidth when we increase the upload bandwidth. We observe that the peers with very low upload bandwidth can only get approximately 65% download bandwidth of configuration level. Even when we actually set the configuration level of the download bandwidth at a higher value, the peers with low upload bandwidth cannot have better download bandwidth. The reason is that when we utilize the incentive model in the P2P system, if a peer constraint its upload bandwidth at a low level and meanwhile it is downloading content from its neighbors, its credit level will finally fall behind others. When its neighbors receive its demand requests for downloading chunks, these neighbors will check its credit level by querying at the bank server. If this peer's credit level is not in the expected range, its neighbors could possibly deny its request. As a result, not matter how this peer configures its download bandwidth. Its actual download bandwidth is actually relative with its

upload bandwidth. If a peer constraints its upload bandwidth at a low level, it cannot get a better download bandwidth. When we increase the upload bandwidth constantly, the actual download bandwidth stays at a constant level when the upload bandwidth gets to a specific level. In this simulation, when a peer's upload bandwidth can get to 24 chunks per time slot, it is able to get a full configured download bandwidth. It indicates that a peer should keep a good upload-download ratio in order to get an optimal download bandwidth (configured download bandwidth).

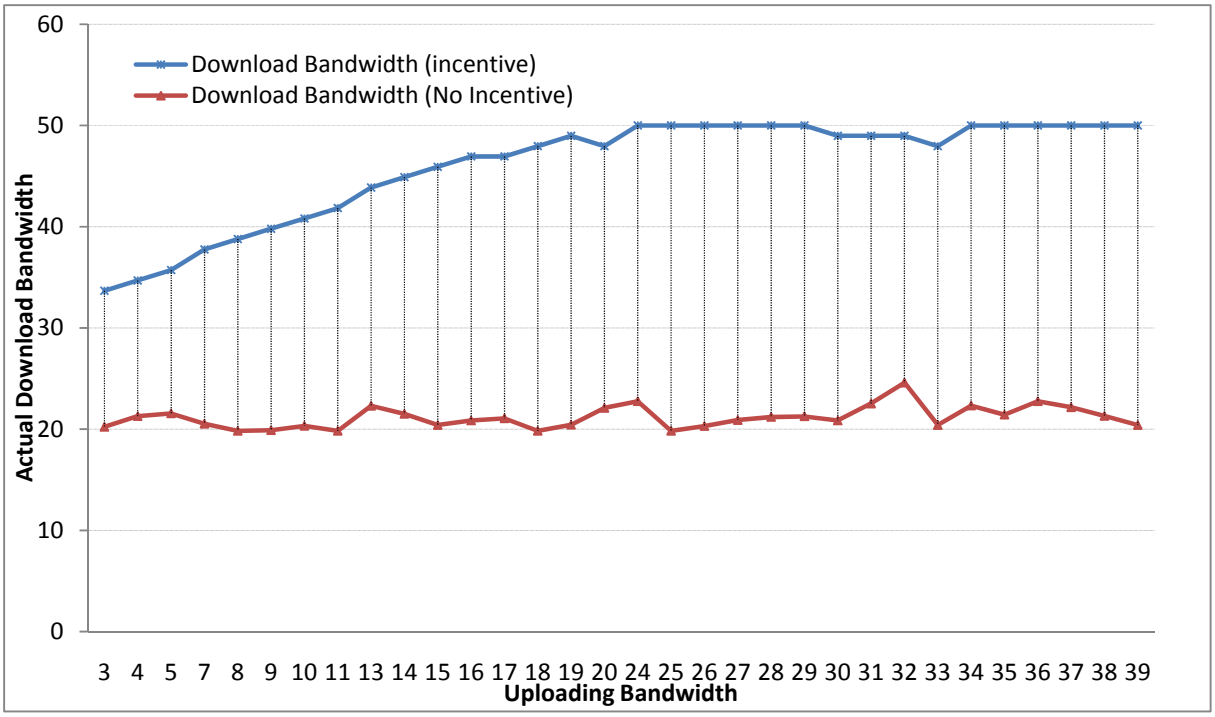


Figure 4.6.1.-2, Actual download bandwidth with incentive and without incentive

Figure 4.6.1-2 shows the difference between a P2P live media streaming system with and without incentive model. In this figure we observe that, if we utilize the incentive model in networks, peers are able to get to their optimal download bandwidth (100% configured download bandwidth) if they are willing to increase upload bandwidth and keep a good

upload/download ratio. However, if we do not utilize the incentive model in the networks, the actual average download bandwidth of peers can only get to 42% of configured download bandwidth. Even if we increase the upload bandwidth of a peer, this peer is probably not able to get to a good download bandwidth. The reason for the performance degradation can be found in Figure 4.6.1-3. Without incentives peers constraint their contribution (uploading), this results in the decrease of total upload volume, so that the average download bandwidth will decrease. If we do not utilize the incentive model in the networks, there is another degradation: a peer is not able to get appropriate benefits even if it has a good contribution, since without incentives the system cannot guarantee extra benefits.

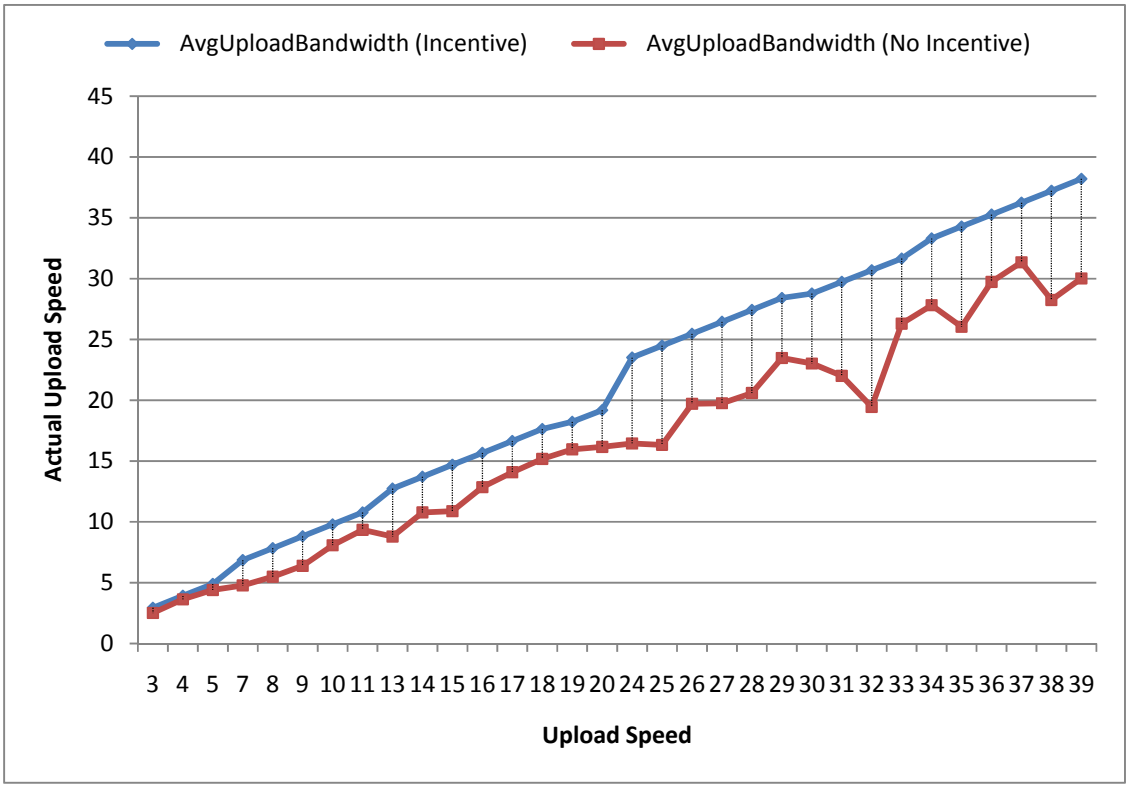
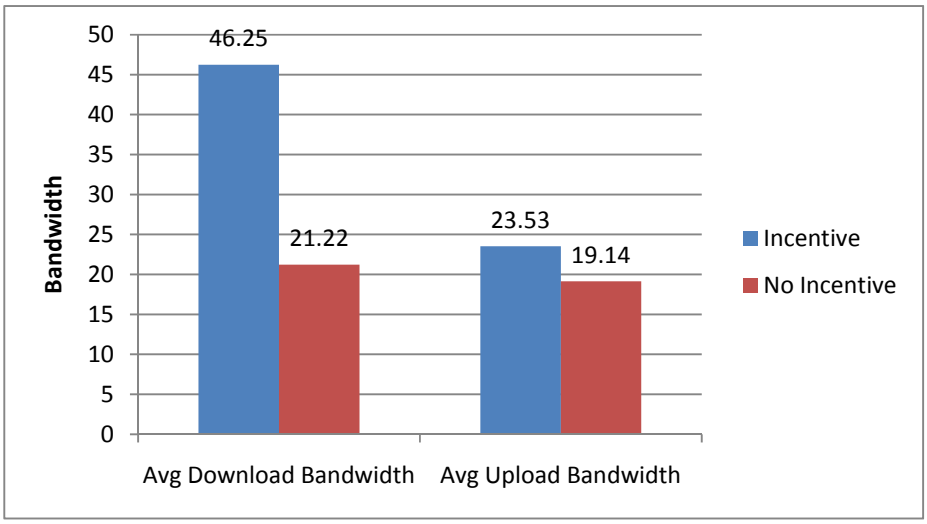
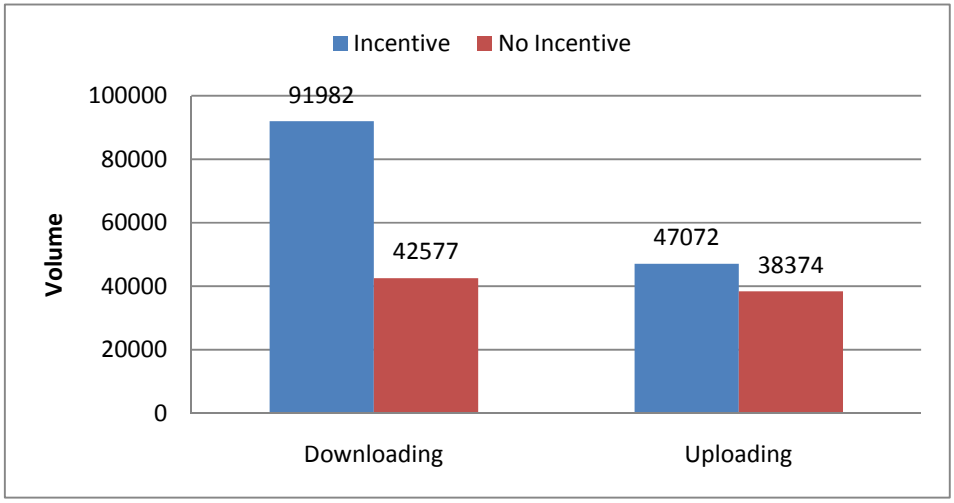


Figure 4.6.1-3, Actual upload bandwidth with incentive and without incentive

Figure 4.6.1-3 shows how the actual upload bandwidth changes in networks with the incentive model and without incentives. We observe that actual upload bandwidth will gradually increase with the increasing of configured upload bandwidth. However, in the incentive model this increase becomes much more rapid. In the figure we observe that, in the incentive model the average upload bandwidth is increased by 23%, and the average download bandwidth is increased by 118%. Since the average download bandwidth is increased, the average startup latency can be reduced by 65%, which is shown in Figure 4.6.1-4.





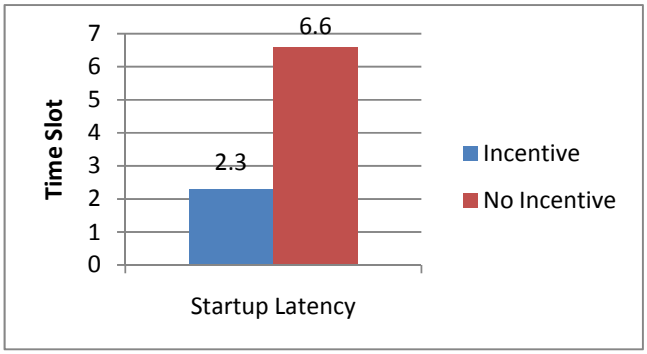


Figure 4.6.1-4 Comparison in incentive mode and non-incentive mode

From the results of this simulation, we can see that the actual download bandwidth and startup latency is highly related to the upload bandwidth in the incentive model. Because a peer with higher upload bandwidth has a better credit level, its download bandwidth is higher. This guarantees that the peer has low startup latency.

4.6.2 Impact of Download Bandwidth

In this simulation, we study the impact of download bandwidth, which leads us to simulate a group of peers with the same upload bandwidth but variable download bandwidth configuration.

Figure 4.6.2-1 shows the average download bandwidth increases gradually to a peak point. This peak point is highly related to the upload bandwidth. When the actual average download bandwidth get to the peak, and we continue to increase the configured download bandwidth, its actual downloading speed will decrease a little until it stays at a constant level. It means the peer will stay in the constantly balanced state. In this state, download bandwidth is balanced with the upload bandwidth.

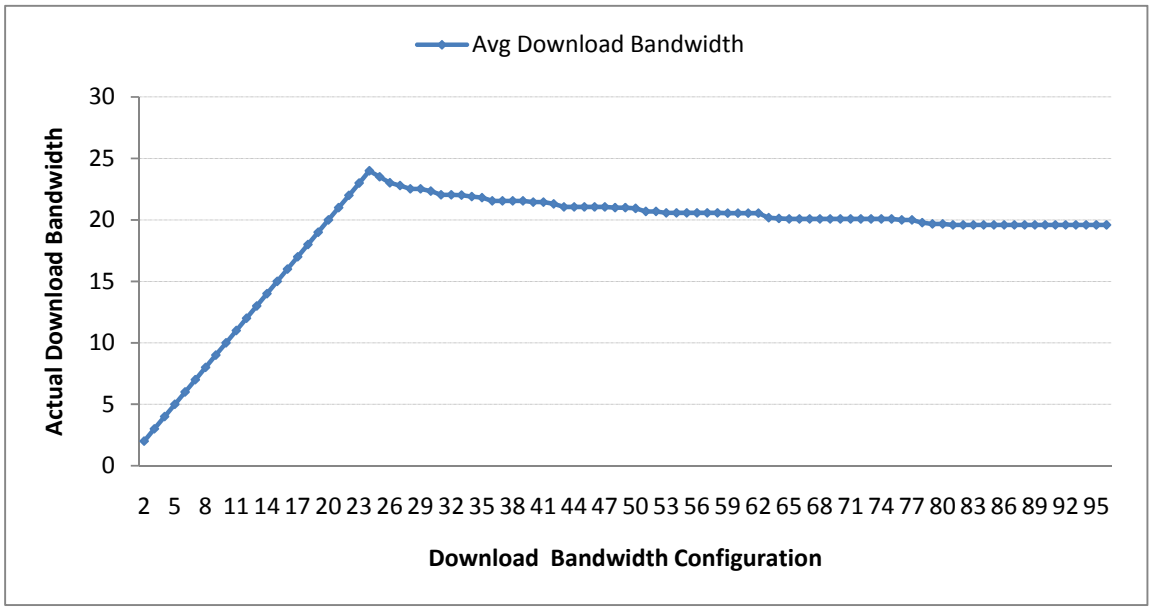


Figure 4.6.2-1, Actual download bandwidth in different configuration

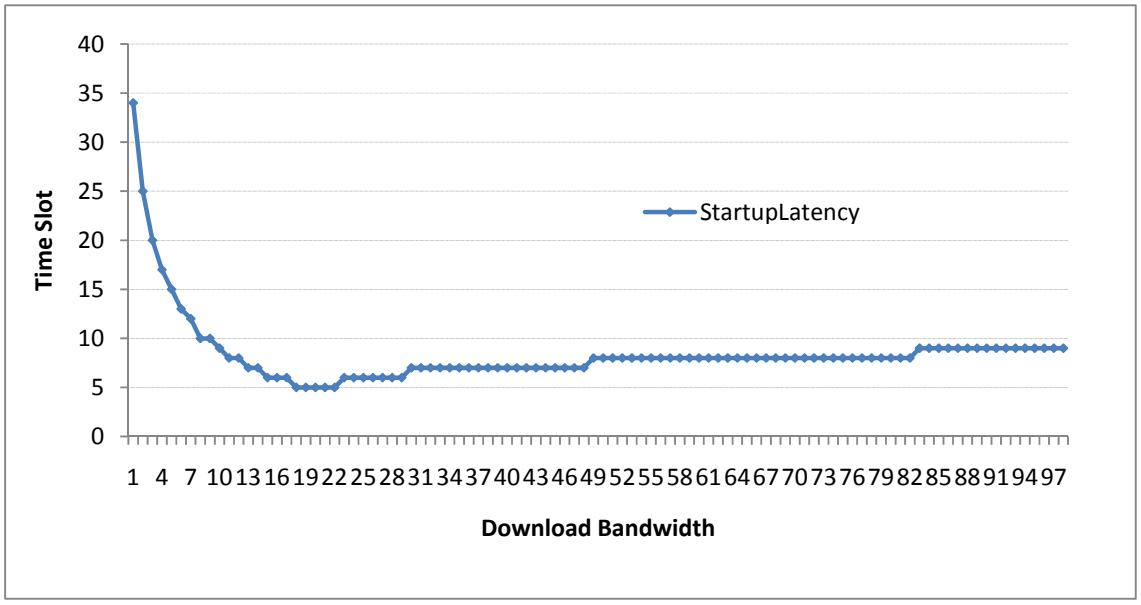


Figure 4.6.2-2, Startup latency with different download bandwidth

Figure 4.6.2.-2 shows that, when the download bandwidth is increasing, a peer's startup latency is decreasing rapidly. However, there is a constraint for the rapid decrease of startup latency. The constraint is the upload bandwidth. If the peer does not increase the upload bandwidth, it cannot further improve its startup latency. Finally the startup latency will be kept at a balanced level which is highly relative with a peer's upload bandwidth.

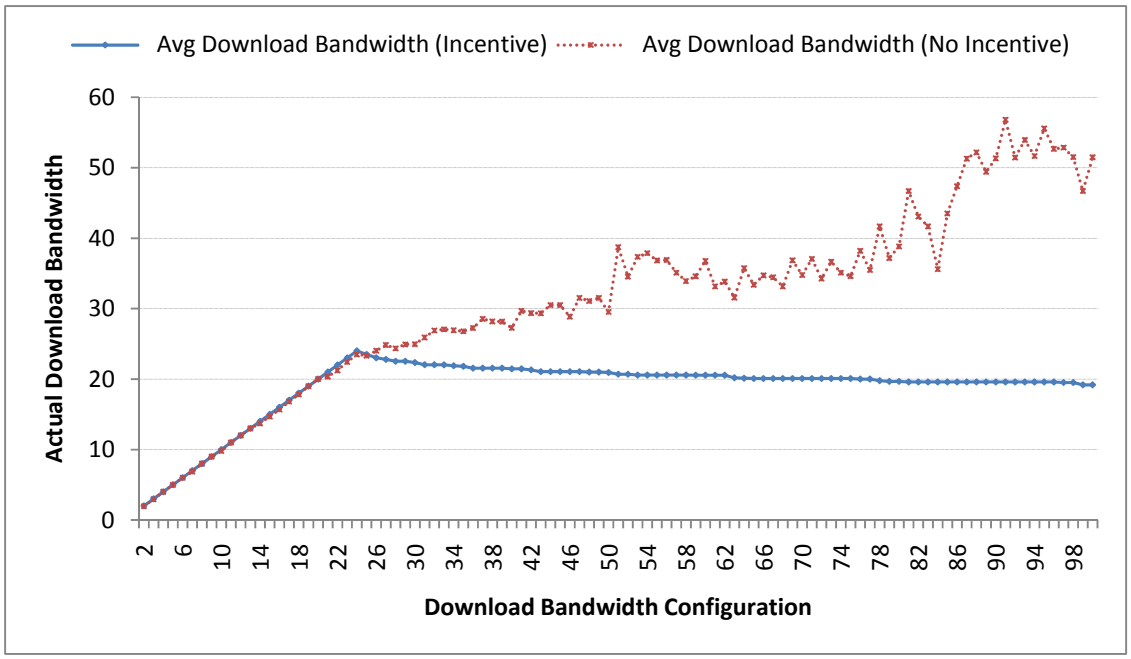


Figure 4.6.2-3, Actual download bandwidth with incentive and without incentives

Figure 4.6.2-3 shows how the actual download bandwidth changes with the incentive model and without incentives. We observe that, in both modes the actual download bandwidth increased gradually and rapidly. In the incentive mode, the download bandwidth will get to a balanced state, in which a peer's download bandwidth is highly relative with its upload bandwidth. If the peer constraint its upload bandwidth, its download bandwidth will be constrained too. However, in the No-Incentive Mode, no matter how we configure a peer's upload bandwidth, we observe that a peer's download

bandwidth will always gradually increase if we increase its configuration of download bandwidth. That means a peer can get a higher download bandwidth with a very low upload bandwidth, which can easily result in free-rider behavior. If this bad upload/download ratio is allowed in a P2P networks, the performance of many peers would be impacted.

4.7 Summary

In this chapter, we introduced an innovative incentive model, a bank incentive model. This model effectively encourages peers to make more contributions and improves the scalability of the P2P live streaming system. This is the most important goal we propose this model for. Table 4.7 summarizes the functionalities of this model.

Streaming Performance	E-commerce Functionalities	Flexibility and Scalability
Query ACK and Deny Priority in Service Queue Advanced Search Pull-Push Helpmate Peer Group selection Bonus and Reward	Advanced search broadcast personal media Ads in buffering VIP movie Support third-party application File conversion for multiple devices	Support multiple platform Banking Credit strategy is scalable

Table 4.7 Summary of the Bank Incentive Model

Our simulation demonstrates that our bank incentive model can solve free-rider problems, apply a balance between a peer's upload bandwidth and download bandwidth, and encourage contributions by utilizing and managing credit level of the peers. Besides the improvement of scalability, our model is designed to support multiple platforms and the extension of incentive strategies.

Chapter 5

Conclusion and Future Work

In this work, we studied the methods for improving transmission efficiency and scalability in the P2P live streaming system.

For the goal of improving transmission efficiency, we present the chunk scheduling algorithms and delivery methods which manage the requesting and transmission of chunks respectively. Furthermore, we proposed an incentive model named bank incentive model to improve the scalability for P2P live streaming system. For the scheduling algorithms which are considered as the core of data-driven protocols, Greedy has the best performance on reducing startup latency while Rarest First is much better in improving streaming continuity. Instead of Mixed which is a combination of Greedy and Rarest First, we propose a new chunk scheduling algorithm named Alternate algorithm that integrates the advantage of Greedy and Rarest First and presents a better comprehensive performance in both startup latency and continuity.

Besides the improvement of chunk scheduling, we also study chunk delivery method to improve the transmission efficiency, which include the evaluation and comparison of Pull method and Push method. Our simulation shows that Push delivery method is much more powerful than pure Pull method, and it can greatly reduce the latency and increase the playback continuity.

Inspired by a common idea of establishing reputation for a peer based on its contribution and consumption, we propose an incentive model to encourage contribution especially for

P2P live streaming system as the major incentive purpose besides ensuring fairness in traditional P2P file sharing, which is named bank incentive model. Our incentive model demonstrates a good performance and efficiency in solving free-rider problems, encouraging peers for more contribution, improving system's scalability, and encouraging peers for broader online activities. In future, a challenge for our bank incentive model, the collusion problem needs to be solved.

References:

[1] Jani Suomalainen, Anssi Pehrsson, and Jukka K. Nurminen. A security analysis of a P2P Incentive mechanism for mobile devices. The Third International Conference on Internet and Web Applications and Services (ICIW 2008), 2008.

[2] International workshop on peer-to-peer systems (IPTPS). 2008 & 2009 http://www.iptps.org/papers.html

[3] "PPLive", http://www.pptv.com/

[4] "PPStream", http://www.ppstream.com/

[5] Bin Yu and Munindar P. Singh. Incentive mechanism for peer-to-peer systems. In Proceedings of the 2nd International Workshop on Agents and Peer-to-Peer Computing, 2003

[6] Michal Feldman, Kelvin Lai, Ion Stoica, and John Chuang. Robust incentive techniques for peer-to-peer networks. ACM E-commerce Conference (EC'04), May

[7] Bridge Q. Zhao, John C.S. Lui, and Dah-Ming Chiu. Analysis of adaptive incentive protocols for P2P networks. In Proceedings of IEEE INFOCOM 2009

[8] Matt Welsh, David Culler, and Eric Brewer. SEDA: An architecture for well-conditioned, scalable Internet Services. In Proceedings of the Eighteenth Symposium on Operating Systems Principles (SOSP-18), 2001

[9] Yipeng Zhou, Dah-Ming Chiu, John C.S Lui. A simple model of analyzing P2P streaming protocols. International Conference on Network Protocols (ICNP), In Proceedings of IEEE 2007

[10] Yang-hua Chu, Sanjay G. Rao, Srinivasan Seshan, and Hui Zhang. A case for end system multicast. In Proceedings of ACM SIGMETRICS, June 2000.

[12] Xinyan Zhang, Jiangchuan Liu, Bo Li, and Tak-Shing Peter Yum. CoolStreaming/DONet: A data-driven overlay network for efficent media streaming. In Proceedings of IEEE INFOCOM'05, March 2005.

[13] Vikash Agarwal and Reza Rejaie. Adaptive multi-source streaming in heterogeneous peer-to-peer networks. In Multimedia Computing and Networking 2005 (MMCN), 2005

[14] Chuang, J.. Designing incentive mechanisms for peer-to-peer systems. Grid Economics and Business Models,

[15] Helix Streaming Server, RealNetworks, https://helixcommunity.org/

[16] Arnaud Legout, G.Urvoy-Keller, and P.Michiardi. Rarest first and choke algorithms are enough. In Proceedings of Internet Measurement Conference 2006

[17] Vinay Pai, Kapil Kumar, Karthik Tamilmani, Vinay Sambamurthy, and Alexander E. Mohr. Chainsaw: Eliminating trees from overlay multicast. In Proceedings of IPTPS 2005, 2005

[18] Darwin Streaming Server, Apple, http://developer.apple.com/

[19] youtube, http://www.youtube.com/

[20] Meng Zhang, Jian-Guang Luo, Li Zhao, and Shi-Qiang Yang. A peer-to-peer network for live media streaming using a push-pull approach. In Proceedings of ACM Multimedia 2005, November 2005

[21] Nazanin Magharei and Reza Rejaie. Prime: peer-to-peer receiver-driven meshbased streaming. In Proceedings of IEEE INFOCOM 2007

[22] Stephen E. Deering and David R. Cheriton. Multicast routing in datagram Internet works and extended LANs. ACM Transactions on Computer Systems, 1990, 8(2):85–110.

[25] Ayman EL-Sayed, Vincent Roca, INIRIA Rhone-Alpes, and Laurent Mathy. A survey of proposals for an alternative group communication service. IEEE Network, 2003. 46–51.

[26] Paul Francis. Yoid: extending the internet multicast architecture. http://www.icir.org/yoid/. 2006.

[27] Dimitrios Pendarakis and Sherlia Shi. ALMI: an application level multicast infrastructure. In Proceedings of the 3rd USNIX Symposium on Internet Technologies and Systems (USITS), 2001.

[28] Suman Banerjee, Bobby Bhattacharjee, and Christopher Kommareddy. Scalable application layer multicast. In Proceedings of ACM Sigcomm 2002, 2002.

[29] Venkata N. Padmanabhan, Helen J. Wang, and Philip A. Chou. Distributing Streaming Media Content Using Cooperative Networking. In Proceedings of ACM NOSSDAV 2002, Miami Beach, FL, USA, 2002.

[30] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony Rowstron, and Atul Singh. SplitStream: high-bandwidth multicast in cooperative environments. In Proceedings of Symposium on Operating Systems Principles (SOSP) 2003

[31] Napster: http://www.napster.com/.

[32] Napster: http://en.wikipedia.org/wiki/Napster.

[33] Yi Cui and Klara Nahrstedtt. Layered peer-to-peer streaming. In Proceedings of NOSSDAV, June 2003

[34] Yi Cui, Baochun Li, and Klara Nahrstedt. oStream: asynchronous streaming multicast in application-layer overlay networks. IEEE Journal of selected areas in Communications (JSAC) special issue on recent advances in service overlay, 2004, 22(1)

[35] Cheng Huang, Jin Li, and Keith W. Ross. Can Internet video-on-demand be profitable?. In Proceedings of ACM SIGCOMM, 2007, 133-144

[36] George Iosifidis and Iordanis Koutsopoulos. Reputation-assisted Utility Maximization Algorithms for P2P Networks. 16th International Workshop on Quality of Service, pp. 20-29, Jun. 2008.

[37] "Emule", http://www.emule-project.net

[38] "Kazaa", http://www.kazaa.com/

[39] Sonja Buchegger and Jean-Yves Le Boudec. A Robust Reputation system for P2P and mobile ad-hoc networks. Technical Report IC/2003/50, EPFL-IC-LCA, 2003.

[40] Sergio Marti and Hector Garcia-Molina. Limited reputation sharing in P2P systems. In Proceedings of ACM 2004 (EC 04), ACM Press, 2004, pp. 91–101.

[41] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. Incentives for combating freeriding in P2P Networks, Technical report, Stanford University, 2003.

[42] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia Molina. The eigentrust algorithm for reputation management in P2P Networks. In Proceedings of the Twelfth International World Wide Web Conference, Budapest, May 2003.

[43] B. Mortazavi and G. Kesidis. Incentive-compatible cumulative reputation systems for peer-to-peer file-swapping.

[44] Thanasis G Papaioannou and George D Stamoulis. Effective use of reputation in peer-to-peer environments. 4th IEEE international symposium on cluster computing and the grid (CCGrid'04), 2004

[45] Yi Hu, Min Feng, Laxmi.N.Bhuyan, and Vana.Kalogeraki. Budget-based self-optimized incentive search in unstructured p2p networks. In Proceeding of IEEE 2009

[46] Alberto Blanc, Yi-Kai Liu, and Amin Vahdat. Designing incentives for peer-to-peer routing. Workshop on Economics of Peer-to-Peer Systems

[47] Emmanuelle Anceaume, Maria Gradinariu, and Aina Ravoaja. Incentive for P2P fair resource sharing. In Proceedings of 5th IEEE international conference on P2P computing (P2P'05), 2005

[48] Bin Yu and Munindar P. Singh. Incentive mechanisms for peer-to-peer systems. In Proceedings of the 2nd International Workshop on Agents and Peer-to-Peer Computing, 2003

[50] Jiangchuan Liu, Sanjay G. Rao, Bo Li, and Hui Zhang. Opportunities and challenges of peer-to-peer Internet video broadcast. In Proceedings of IEEE 2007, Special Issue on Recent Advances in Distributed Multimedia Communications, 2007.

[51] Meng Zhang, Li Zhao, Yun Tang, Jian-Guang Luo, and Shi-Qiang Yang. Large-scale live media streaming over peer-to-peer networks through global Internet. In Proceedings of ACM workshop on Advances in peer-to-peer multimedia streaming (P2PMMS), 2005, 21-28

[52] SaurabhTewari and Leonard Kleinrock. Analytical model for bittorrent-based live video streaming. in IEEE NIME Workshop, Jan. 2007.

[53] Chuan Wu and Baochun Li. Optimal peer selection for minimum-delay peer-to-peer streaming with rateless codes. In Proceedings of ACM Workshop on Advances in P2P Multimedia Streaming, 2005.