Summer 7-2-2012

On heterogeneous user demands in peer-to-peer video streaming systems

Zhipeng Ouyang
University of Nebraska - Lincoln

ON HETEROGENEOUS USER DEMANDS IN PEER-TO-PEER VIDEO STREAMING SYSTEMS

by

Zhipeng Ouyang

A DISSERTATION

Presented to the Faculty of
The Graduate College at the University of Nebraska
In Partial Fulfillment of Requirements
For the Degree of Doctor of Philosophy

Major: Computer Science

Under the Supervision of Professors Lisong Xu and Byrav Ramamurthy

Lincoln, Nebraska

May, 2012

ON HETEROGENEOUS USER DEMANDS IN PEER-TO-PEER VIDEO STREAMING SYSTEMS

Zhipeng Ouyang, Ph.D.

University of Nebraska, 2012

Advisors: Lisong Xu and Byrav Ramamurthy

A Peer-to-Peer (P2P) video streaming system usually consists of a large number of peers, which have heterogeneous physical properties. Orthogonal to the physical heterogeneity, there is another type of heterogeneity called demand heterogeneity. Namely, peers have their own demands on the quality and type of the streaming service. The problem of demand heterogeneity has received little attention and as a result current P2P video streaming systems cannot achieve satisfactory performance due to demand heterogeneity. In this dissertation, we study how to design efficient P2P video streaming systems with heterogeneous user demands.

First, we study the problem of heterogeneous user demands on video quality. We design an efficient P2P live streaming system which greatly reduces the bandwidth consumption and still achieves satisfactory streaming quality. To reduce the bandwidth consumption, we use adaptive streaming, so that the streaming rate of a peer is commensurate with its window size. In order to maintain satisfactory streaming quality even in the case when peers dynamically change their window sizes, small-window peers are designed to contribute part of their bandwidth to help large-window peers.

Second, we study the problem of heterogeneous user demands on playback delay. Specifically, we consider time-shifted streaming service and bounded-delay live streaming service. With time-shifted streaming service, a peer can watch a program

which has been broadcast live before it joins the system. We propose a cooperation-based prefetching design which distributes video segments more widely in a P2P streaming system and thus greatly improves the streaming quality. With bounded-delay live streaming service, some peers (e.g., paid users) can watch the program with short and bounded playback delays, and other peers (e.g., free users) with best-effort short playback delays. We prove that the bounded-delay live streaming problem is NP-Complete, and propose a heuristic algorithm which constructs an efficient P2P overlay structure of peers by considering their playback delay requirements and their physical properties.

The promising results in this dissertation show that our algorithms are effective in meeting heterogeneous user demands on video quality and playback delay. In the future, we are interested in designing efficient P2P streaming systems in mobile and cloud environments.

ACKNOWLEDGEMENTS

I am most grateful to my advisors, Dr. Lisong Xu and Dr. Byrav Ramamurthy. Their guidance and insights over the years have been invaluable to me. I feel especially fortunate for the patience that they have shown with me when I firstly stepped into the field of peer-to-peer streaming. I am indebted to them for teaching me both research and writing skills. Without their endless efforts, knowledge and patience, it would be impossible for me to finish all my dissertation research and Ph.D study. It has been a great honor and pleasure for me to do research under their supervision.

I would like to thank Dr. Hong Jiang, Dr. Mehmet Can Vuran and Dr. M. Cenk Gursoy for serving as my Ph.D committee members and reviewing my dissertation. I also owe thanks to Jie Feng, Miao Wang and Peng Yang, who helped me with various problems.

Before coming to the University of Nebraska-Lincoln, I spent seven years in East China Normal University, Shanghai, China, where I decided to work on computer networking. I thank Dr. Fuke Shen for introducing me the field of computer networking and helping me complete my thesis on computer networks.

I must thank my fiancee Wei Zeng and my family members in China, who support me a lot both emotionally and financially. Without their endless love and encouragement I would have never completed this dissertation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Peer-to-Peer (P2P) Streaming Systems

Peer-to-peer (P2P) streaming systems, in which individual users (also called peers)[1] collaboratively distribute video streams, have become one of the most important applications on the Internet today [40, 38, 84]. The basic idea of P2P streaming systems is to encourage peers to contribute their upload bandwidth to forward video streams to other peers when downloading their video streams. P2P streaming systems reduce server bandwidth consumption, and thus can support a large number of users.

P2P streaming systems can be further divided into two different types: P2P live streaming [44] and P2P video-on-demand (VOD) streaming [21]. In a P2P live streaming system, users can watch real-time videos whenever they join the system. They can choose which video to watch by subscribing to different program channels, but they cannot jump backward or forward when watching the video. In a P2P VOD streaming system, users can choose to play pre-recorded videos by searching using the interface provided by the P2P software. The pre-recorded videos are stored in

[1] We hereinafter use the terminologies *user* and *peer* interchangeably to refer to an Internet user in a P2P streaming system.

the VOD library, and can be served at anytime. During the playback, users can jump forward, jump backward, pause and stop.

P2P streaming systems have attracted lots of interests from both academia and industry. The academic research mainly focuses on optimal solutions in terms of service quality and resource efficiency (e.g., providing smooth video streams in case of peer churn [46, 66], minimizing the video playback delay [4, 63], optimizing peers' bandwidth consumption [94, 2], etc.), and these efforts derive theoretical performance bounds for P2P streaming systems. On the other hand, commercial P2P streaming systems (e.g., UUSee [70], PPLive [58], Sopcast [64]) have been widely deployed and have attracted a large number of users. They can successfully serve thousands of video streams to millions of P2P users with low server bandwidth consumption.

The work described in this dissertation is motivated by the observation that users in a P2P video streaming system may have different demands on the type and quality of streaming service. The rest of the dissertation is organized as follows. Chapter 1.2 explains heterogeneous user demands in P2P streaming systems. Chapters 1.3 and 1.4 describe two types of heterogeneous user demands: heterogeneous video quality demands and heterogeneous playback delay demands, respectively. Finally, Chapter 1.5 summarizes the contributions of the dissertation.

1.2 Heterogeneity in P2P Streaming Systems

In a P2P streaming system, peers usually have heterogeneous physical properties, such as various bandwidth capacities, widely-spread geographical locations, and diverse viewing appliances. We refer to the existence of these various physical properties as physical heterogeneity. Most of the current P2P streaming research [36, 72, 41, 54] tackles the physical heterogeneity problem, and studies how to provide satisfactory

streaming quality to users with heterogeneous physical properties.

There is another important type of heterogeneity called demand heterogeneity, and it is orthogonal to the aforementioned physical heterogeneity. Namely, users have their own demands on the quality and type of the streaming service. For example, some users in a P2P streaming system may watch a video with higher playback resolutions and shorter playback delay than others. That is, users may have different demands on video quality and delay. The demands of users could be independent of their bandwidth capacities, locations and other physical properties. While the problem of physical heterogeneity has been extensively studied for P2P streaming systems, the problem of demand heterogeneity however has received little attention. As a result, the current P2P streaming systems cannot provide satisfactory streaming quality to users with heterogeneous demands.

However, it is challenging to design a P2P streaming system for users with heterogeneous demands. One important reason is that the demands of users are more dynamic than the physical properties of the users. A user usually has a fixed upload capacity, a fixed location (at least in a short time period). However, a user can change his/her demand at anytime. For example, a user can change his/her demand on the video quality at any time. The dynamic demand significantly complicates the design of a P2P streaming system.

In this dissertation, we focus on two types of demand heterogeneity: heterogeneous user demands on video quality discussed in Chapter 1.3 and heterogeneous user demands on playback delay discussed in Chapter 1.3. In the first type, users in a P2P live streaming system may watch the same channel with different window sizes which have different quality demands. In the second type, users in a P2P streaming system may watch the same channel with different playback delay requirements.

1.3 Heterogeneous Video Quality Demands in P2P Live Streaming Systems

What are heterogeneous video quality demands? The video quality of a user is defined as the video playback resolution of the user. Users in a P2P streaming system have heterogeneous video quality demands. For example, different users may watch the same channel with different window sizes. These window sizes have different number of pixels, and the video is displayed at different playback resolutions. Specifically, a user with a large window has a demand for high video quality, whereas a user with a small window has a demand for low video quality.

Why is it important to study the problem of video quality demand heterogeneity? This is because if we can solve this problem, we can significantly reduce the bandwidth consumption of a P2P streaming system, which is a serious concern of current P2P streaming systems and is strongly complained by Internet Service Providers (ISPs). Current P2P streaming systems send the same encoded video data at the same streaming rate to all users watching the same channel, regardless of their heterogeneous demands on video quality. We propose to send different encoded video data at different streaming rates to users with different demands on video quality. For example, we can send more encoded video data at a high streaming rate to users with a demand for high video quality, and send less encoded video data at a low streaming rate to users with a demand for low video quality. By doing so, we can significantly reduce the total amount of consumed bandwidth, especially for a large P2P streaming system supporting millions of users.

Why is it challenging to tackle the video quality demand heterogeneity problem? There are two major reasons. First, one important reason that current P2P streaming systems send the same encoded video data at the same streaming rate to all users

regardless of their demands on video quality is that it greatly simplifies the design of P2P streaming systems. By sending different encoded video data at different streaming rates to users with heterogeneous video quality demands, we can significantly reduce the total bandwidth consumption, however, it also considerably complicates the design of a P2P streaming system. Second, a user may dynamically change its demand on video quality at any time, however, it is challenging to quickly meet the new demand of a user without disturbing other users in the system. We have successfully solved all above challenges, and our solutions are described in Chapters 3 and 4.

1.4 Heterogeneous Playback Delay Demands in P2P Streaming Systems

What are heterogeneous playback delay demands? The playback delay of a program experienced by a user is defined as the time interval between the time when the program is live broadcast and the time when the user finally watches the program. Users in a P2P streaming system have heterogeneous playback delay demands, because they want to watch programs at their convenient times and different users may have different convenient times. For example, some users may want to watch a program when it is broadcast live, whereas some users may want to watch it at a later time.

Why is it important to study the problem of playback delay demand heterogeneity? This is because current P2P streaming systems provide users with very limited playback delay choices: either best-effort live streaming or VOD streaming, and thus cannot meet the diverse needs of users.

- With best-effort live streaming, a P2P streaming system tries its best effort to stream a live broadcast program to its users as quickly as possible, but there

is no guarantee on the playback delay. The playback delay of current P2P best-effort live streaming systems is in the range of seconds to minutes.

- With VOD streaming, a P2P streaming system streams the pre-recorded video of a program to its users, and thus the VOD streaming service becomes available to its users only after the program has finished and been recorded. Therefore, the playback delay of current P2P VOD streaming systems is very long and is in the range of hours to days.

In this dissertation, we study how to meet the diverse needs of users by providing more playback delay choices.

- Time-shifted streaming: This type of streaming service lies between live streaming and VOD streaming. With live streaming, the playback delay is relatively short and in the range of seconds to minutes (i.e., soon after a program has been live broadcast). With VOD streaming, the playback delay is relatively long and in the range of hours to days (i.e., after a program has finished and recorded). With time-shifted streaming, the playback delay is in the middle and in the range of minutes to hours. With time-shifted streaming, a user can watch a program at a time later than its scheduled live streaming time but before the time when the VOD streaming service becomes available, and thus provides users with more flexibility on the playback delay. In addition, a user can jump to any video segment within the time-shifted period predetermined by the service provider. For example, in a time-shifted streaming system with a 2-hour time-shifted period, a user can watch any video segment broadcast in the past 2 hours.

- Bounded-delay live streaming: With such a type of streaming service, a P2P streaming system streams a live broadcast program to some users (e.g., paid

users) with a short and bounded delay, and to other users (e.g., free users) with a best-effort delay. While the playback delay of best-effort live streaming users is in the range of seconds to minutes, the playback delay of bounded-delay live streaming users is in the range of seconds. More importantly, the playback delay of best-effort live streaming users is not guaranteed whereas the playback delay of bounded-delay live streaming users is bounded and guaranteed.

Why is it challenging to tackle the playback delay demand heterogeneity problem? It is challenging to design a time-shifted streaming system, because users in such a system may watch different video segments in the time-shifted period, and thus it is hard for different users to share their video segments and is hard for the system to achieve satisfactory streaming quality. It is challenging to design a bounded-delay live streaming system, because many factors contribute to the playback delay experienced by a user, and thus it is hard to control the playback delay especially in a system with a large number of dynamic users with heterogeneous physical properties. We have successfully solved all above challenges, and time-shifted streaming is described in Chapter 5 and bounded-delay live streaming is described in Chapter 6.

1.5 Our Contributions

Figure 1.1 shows the relationship among different chapters, where the root topic in this dissertation is demand heterogeneity in P2P streaming systems. Among various types of demand heterogeneity, we focus on demand heterogeneity on video quality and playback delay. Chapter 2 presents the related work. Chapters 3 and 4 can be read together, in that Chapter 3 explores the design of simple and efficient systems dealing with heterogeneous demands on video quality, and Chapter 4 formulates the problem as a demand-supply problem, and studies the problem optimization. Chapter 5 and

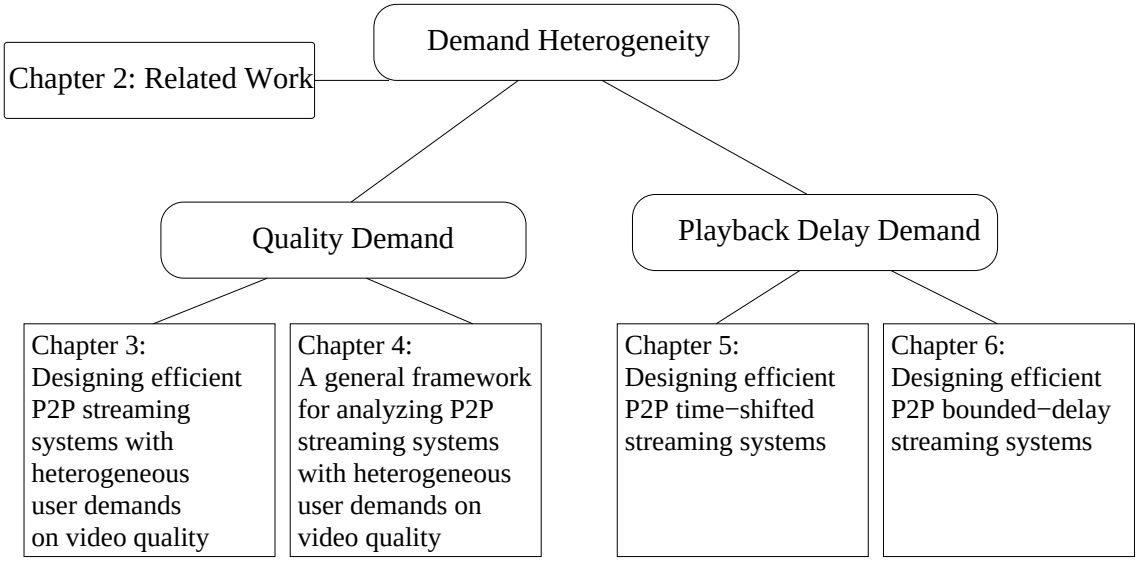

Figure 1.1: Disseration Organization and Chapter Relationship.

Chapter 6 focus on demands on playback delay, but they can be read independently. Chapter 5 investigates P2P time-shifted streaming and Chapter 6 presents the design of P2P bounded-delay live streaming.

The contribution of Chapter 3 is that we propose two simple and efficient schemes for P2P live streaming systems with heterogeneous video quality demands, with the goal of significantly reducing the total bandwidth consumption while still maintaining satisfactory streaming quality. Specifically, we design and study two schemes which represent two different design philosophies: a well-balanced and more distributed method called Partial Forwarding Scheme (PFS), and a supernode-based method called Partial Participation Scheme (PPS).

The contribution of Chapter 4 is that we propose a general theoretical framework for analyzing P2P streaming systems with heterogeneous user demands on video quality. The difference between Chapters 3 and 4 is that Chapter 3 considers only a specific video encoding method which simplifies the system design, whereas Chapter 4 considers a more general video encoding method and then extends the work proposed

in Chapter 3 to a more general system design.

The contribution of Chapter 5 is that we design an efficient P2P time-shifted streaming system. The difficulty lies in that users in such a system may watch different video segments in the time-shifted period and thus it is hard for different users to share their video segments and is hard for the system to achieve satisfactory streaming quality. We propose a cooperation-based prefetching design which distributes video segments more widely in the system and thus makes it easier for users to share their video segments.

The contribution of Chapter 6 is that we design an efficient P2P bounded-delay live streaming system. The difficulty lies in that many factors contribute to the playback delay experienced by a user, and thus it is hard to control the playback delay especially in a system with a large number of dynamic users with heterogeneous physical properties. We first show that the bounded delay live streaming problem is NP-complete, and then propose a heuristic algorithm called high fanout promotion (HFP) algorithm. Intuitively, HFP constructs an efficient P2P overlay structure of users by considering their different playback delay requirements, their upload bandwidth capacities and their distances to the streaming server.

The work described in Chapter 3 has been published in [52, 89]. The work described in Chapter 4 has been published in [90]. The work described in Chapter 5 has been published in [91]. The work described in Chapter 6 is is currently under review [92].

Chapter 2

Related Work

2.1 Related Work on P2P Video Streaming Systems

P2P video streaming systems have been widely used in large-scale video distribution, and there are lots of studies on P2P video streaming systems. In a P2P video streaming system, peers are organized into an overlay, where they maintain a set of connections to other peers. Generally, P2P video streaming systems have tree-based, multi-tree-based and mesh-based overlay structures [40]. In a tree-based overlay [28, 71], the video streaming server serves as the root of the tree and peers are located at different levels of the tree. Peers receive video contents from their parent peers. This structure is not resilient to peer churn and leaf peers' bandwidth cannot be utilized. Multi-tree based systems are proposed in [6, 66, 46], where the video stream is divided into substreams and each substream is delivered over its corresponding subtree. So the leaf peers' upload bandwidth can be used by other peers. The mesh-based overlay is proposed to improve system robustness and bandwidth utilization [97, 30, 95]. When a peer joins a mesh-based overlay, it first contacts the tracker

server, which maintains the information of online peers. Then the server returns a list of online peer IDs (e.g., IP address) to the newly joined peer. Then this peer selects some peers from the list and sets up connections with them. Since the mesh-based overlay is resilient to peer churn and has low implementation complexity, it is widely used in commercial systems [58, 60, 70], and we also adopt it in our dissertation.

To improve the throughput of P2P video streaming systems, several content scheduling methods have been proposed, such as a stream-level method [11] and a block level method [94]. The former uses layered coding and determines routes for streams of different layers, while the latter is more fine-grained because it describes how a node fetches a particular block from a particular neighboring peer. [96] proposes random scheduling, which is simple and has high performance with the proper configuration. Adaptive queue based chunk scheduling [19] and min-cost scheduling [95] study the optimal scheduling to fully utilize the resources and to achieve the maximum streaming rate. [93] formulates the P2P live streaming as a network flow problem and maximizes the overall performance. [47] proposes a distributed algorithm called "random useful packet forwarding" and theoretically proves that it is optimal.

Recently, cloud service attracts lots of efforts of leveraging it to support video streaming from both industry [50, 14] and academia [26] since it provides an agile and scalable resource access. [26] proposes a cloud-based video proxy that delivers high-quality streaming videos using a scalable codec to adapt to network dynamics. In [82], Wu *et al.* check the fundamental question of how to configure the cloud utility to meet the highly dynamic demands, and address this practical issue using VoD as the example application. [65] finds that clouds are ready to support individual users, but many efforts are required to enable clouds to support large distributed applications. [73] is the very first work on migrating live media streaming to a cloud platform, and

presenting the results of combining cloud service and P2P live streaming. There are still many areas of future research to be explored to adopting cloud service in P2P video streaming.

2.2 Related Work on Heterogeneity in P2P Video Streaming Systems

P2P video streaming systems consist of thousands of millions peers which have various physical properties, e.g., their access bandwidth, widely-spread geographical locations, diverse viewing appliances and so on. The heterogeneous properties have already been considered in the P2P video streaming system design. [10] proposes the framework which models differentiated services to users with heterogeneous physical properties. [39] shows that peer bandwidth heterogeneity can significantly increase the delay performance of all peers, and the snow-ball streaming algorithm is proposed for minimizing the delay bound in realtime P2P video streaming. A new scheduling approach for layered video streaming, called LayerP2P[83] considers various link bandwidth in a P2P overlay. It achieves high throughput, high layer delivery ratio, low useless packets ratio, and low subscriber jitter. Peers in a P2P video streaming system may have different Internet service providers, which may lead to inefficient network resource usage and/or low application performance. In [85], Xie *et al.* propose a simple architecture called P4P to provide more effective cooperation based traffic control between applications and network providers.

Different devices in P2P video streaming systems are considered in [35, 36]. Since some devices may have limited computing power, an additional metadata overlay is constructed to minimize the total computing overhead. [35] also leverages active peer cooperation without demanding infrastructure support. In [36], Liu *et al.* study how

to efficiently serve heterogeneous devices with limitations different from those of desktop computers. They propose the Dynamic Bi-Overlay Rotation (DOOR) scheme, and design the corresponding rotation scheme that reacts to dynamic situations and balances across multiple types of resources on individual nodes. They extend their work in [37] by studying the upper bound of achievable performance with the available resources.

Orthogonal to the aforementioned physical heterogeneity, there is another type of heterogeneity called demand heterogeneity. Namely, peers have their own demands on the quality and type of the video streaming service. However, the problem of demand heterogeneity has received little attention, and there is no much related work. In [77], different P2P sessions are treated with different priorities, and a novel application layer approach called Diverse is presented. Different from their work, our work focuses on different services at the peer level. In this dissertation, we consider two types of user demands: first, demands on the video quality, and second, demands on the playback delay. The related work is discussed in following two sections.

2.3 Related Work on Heterogenous User Demands on Video Quality

Even though P2P video streaming systems have been extensively studied, to the best of our knowledge, the topic of support users' heterogeneous demands on video quality has not been addressed before and there is no related work on this topic. Below, we briefly summarize the related work on adaptive streaming and on cooperation schemes.

2.3.1 Related Work on Adaptive Streaming

Several video coding techniques, including single-layer coding, layered coding, and MDC, have been studied for P2P video streaming systems. Compared to layered coding and MDC, single-layer coding is the simplest to implement and has the highest coding efficiency. Layered coding and MDC have been studied for P2P video streaming systems to address the bandwidth heterogeneity [83, 11, 55, 45, 5] or to provide incentives [43, 49]. However, they are still not widely used in commercial P2P video streaming applications mainly because of their complexity and relatively low coding efficiency compared to single-layer coding. As the gap between the coding efficiency of single-layer coding and layered coding has been significantly reduced recently, we expect to see an increased use of layered coding. For example, H.264/SVC (layered coding) [76] achieves the same video quality as H.264/AVC (single-layer coding) at the cost of only 10% rate increase. Different from the existing work on adaptive streaming which addresses bandwidth heterogeneity or incentive issues, we use adaptive streaming to reduce the bandwidth consumption in case of heterogeneous user demands on video quality.

2.3.2 Related Work on Cooperation Schemes

The concept of helpers is introduced in [57] for P2P based file sharing systems. Several cooperation schemes [67, 12, 34, 24] have been proposed so that a user can contribute its unused upload bandwidth to another channel not being watched by the user itself. These cooperation schemes can improve the overall system performance when some channel has excessive upload capacity whereas another channel has insufficient upload capacity. To tackle the channel churn and channel resource imbalance problems in a multi-channel P2P streaming system, View-Upload Decoupling (VUD) [81]

decouples user downloading from uploading and thus a user can contribute its upload bandwidth to any channel that the user may or may not be viewing. Different from these cooperation schemes which consider cooperation among different channels, our work considers cooperation within a single channel.

Peer-Assisted Transcoding (PAT) [35] also considers cooperation within a single channel. However, it studies the cooperation among users using heterogenous types of devices with different screen sizes, different color depths, and different bandwidth capacities. But our work studies the cooperation among users with different window sizes. A fundamental difference between their work and our work is that the type of a device is fixed whereas the size of a window is dynamic.

2.4 Related Work on Heterogeneous User Demands on Playback Delays

We first review related work on different types of streaming services providing different playback delays, and then review live streaming systems providing differentiated playback delays.

2.4.1 Related Work on Different Types of Streaming Services Providing Different Playback Delays

Most studies on P2P video streaming systems focus on live streaming [44, 31, 19] and Video-on-Demand (VOD) streaming [21, 40, 42]. [94, 93] use the mesh structure, and formulate P2P live streaming as a network flow problem to maximize the overall performance. [94] proposes the live streaming system design using layered coding in heterogeneous networks. "random useful packet forwarding" is proposed in [47] and

is proved to be optimal. In [31], a live stream is divided into serval substreams, and it uses a hybrid pull-push method which reduces the playback delay and the control overhead. [25] claims that it is more challenging to design P2P VOD streaming because of less synchrony among VOD peers sharing video content. [22] analyzes the benefits of P2P VOD streaming and develops an analytical model capturing many critical features of P2P VOD streaming. In [32], iPass is proposed to use a dynamic buffering-progress-based peering strategy for asynchronous streaming. It encourages peers to contribute their upload bandwidth by enabling those of higher contribution to pre-fetch content at a higher speed. [9] introduces the popular Bit-Torrent P2P technology into P2P VOD streaming and proposes Toast to offload work from the video streaming server.

Providing low playback delays to peers in P2P live streaming is studied in [97]. In [62], a mesh-based protocol called Fast-Mesh is proposed to optimize the source-to-peer delay. With this protocol, peers select their parents based on their network powers which are defined as their throughput and delay ratios. In [4], the low playback delay and high quality video is provided through maintaining a set of multicast trees, prioritizing packets and retransmitting locally. In [8], the source sending rate is adapted to the bandwidth availability and the congestion, so that the low end-to-end delay is achieved in a multi-party conference. Liu et al. [39] study the delay constraints and derive the minimum delay bounds for realtime P2P video streaming. They propose a snow ball streaming and show that it archives better delay performance than tree-based streaming. In [63], Qiu et al. present heuristics to solve the minimum mesh delay problem. Minimizing end-to-end delay under dynamic and heterogeneous network environments is investigated in [23], which optimizes the bandwidth allocation to achieve the minimum average end-to-end P2P playback delay.

There are some studies on time-shifted streaming. In [18], time-shifted streaming

is used to guarantee video continuity. Peers that suffer service disconnection can join a time-shifted stream to keep the video continuity. [13] proposes that a peer uses one cache to store the content which they are watching, and another cache to store the video segments to implement the time-shifted feature. In [20, 51], the time-shifted feature is implemented by caching a portion of a video stream at each peer and serving it asynchronously. Liu et al. [86] propose peer-assisted time-shifted stream system based on connected devices located at the home which are stable and can be easily tracked. However, these studies have not yet answered the fundamental problem of how peers fetch the specific video segments from the specific neighbor peers so that the overall system performance is optimized.

2.4.2 Related Work on Live Streaming Systems Providing Differentiated Playback Delays

There are research studies on providing differentiated service in P2P streaming systems. Clevenot-Perronnin *et al.* [10] establish a multiclass fluid model to allocate upload bandwidth to different peer groups in a P2P download file systems. Wu *et al.* [77] propose an approach, named *Diverse*, to provide differentiated bandwidth allocation services to different streaming sessions. These studies focus on bandwidth allocation and focus on the overall performance of high priority groups. Different from their works, our work is to deliver bounded delay video to as many subscribers as possible. In ACTIVE [98], active peers dynamically move closer to the root, so that they can achieve low playback delay. However, ACTIVE does not work for our problem due to the following reasons: 1. It only supports a small number of peers with low delay, while P2P streaming systems usually consist of many more peers. 2. Active peers do not necessarily have low delay connections and high upload bandwidth, so moving active peers may not help in serving subscribers. In [29], peers are

divided into several regions according to locality of interests. Frequent updates of peers are propagated under certain time constraints. A latency prediction system is proposed in [3] to help peers cluster themselves. But in our bounded delay problem in this dissertation, since all peers are viewing the same video, the divide and conquer strategy does not work.

Chapter 3

Designing Efficient P2P Streaming Systems with Heterogeneous User Demands on Video Quality

3.1 Motivation and introduction

Peer-to-peer (P2P) streaming systems, in which individual users (also called peers) collaboratively distribute video streams, have become one of the most important applications today [40, 38, 84]. As the population of P2P streaming users grows, the total amount of their consumed bandwidth increases rapidly and has become a great challenge to Internet Service Providers. It is therefore important to design a bandwidth-efficient P2P streaming system, especially for a large number of users.

We have observed that different peers in a P2P live streaming system may watch the same channel with different window sizes which have different quality demands. These window sizes have different number of pixels, and the video is displayed with different playback resolutions. To simplify the description, we use the window size

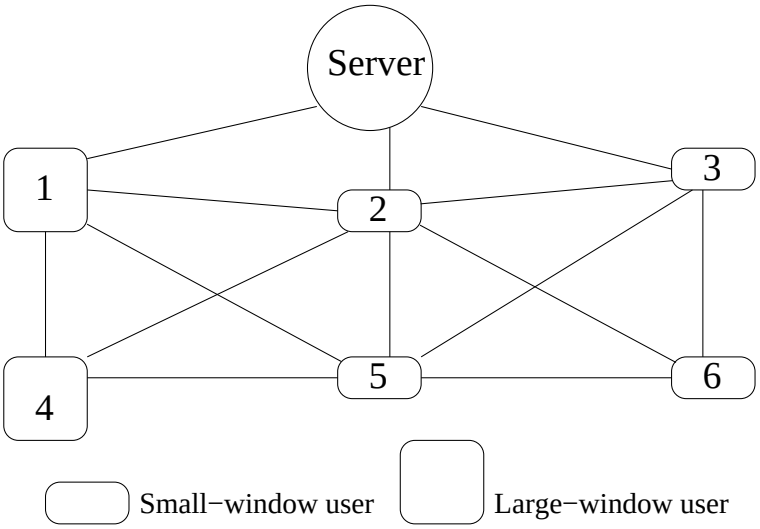

Figure 3.1: Snapshot of a P2P live streaming system, where users watch the same channel with different window sizes.

to represent the corresponding video quality demands. More importantly, they can dynamically change their window sizes at any time through *dynamic window resizing* events.

A user in a P2P streaming system may customize its window size for various reasons, such as reducing the window size to temporarily check other information on the other area of the screen, or increasing the window size for better video quality. Consequently, users watching the same channel may have different window sizes. To simplify our discussion, we consider only two groups of users based on their window sizes: large-window users and small-window users.

Figure 3.1 shows a snapshot of a P2P live streaming system with a mesh overlay, where users 1 and 4 watch the channel in a large window, and users 2, 3, 5, and 6 watch the same channel in a small window. A user can dynamically change its window size according to its own preference. For example, at some time, user 1 may reduce its window size and use the remaining screen to check emails, and change back to a full-screen window after reading emails. At some time, user 6 may change to a

full-screen window because the channel is very interesting. Since a user can change its window size at any time, we refer to these events as *dynamic window resizing* events. Note that due to dynamic window resizing, a large-window user may not have any large-window neighbors. For example, if user 1 reduces its window size from large to small, user 4 does not have any large-window neighbors.

We have the following three design goals for a scheme to support dynamic window resizing.

- *Design Goal 1: Efficient Bandwidth Consumption*: The scheme should effectively reduce the total bandwidth consumption, which can be measured by the total download rate of all users (or equivalently the total upload rate of all users).

- *Design Goal 2: Good Streaming Quality at Resizing Events (i.e., Short Resizing Delay for Small-Window Users)*: The scheme should achieve short resizing delay, which is defined as the delay it takes for a user to see a high-resolution video when the user changes its window size from small to large.

- *Design Goal 3: Good Streaming Quality at Other Times*: The scheme should maintain good streaming quality, which can be approximately measured by the playback continuity [97] that is the ratio of the total number of packets received by a user before the playback deadline to the total number of packets that should be received by the user.

For the second design goal, notice that there is no delay when a user changes its window size from large to small as the user already has all the data required by a small window.

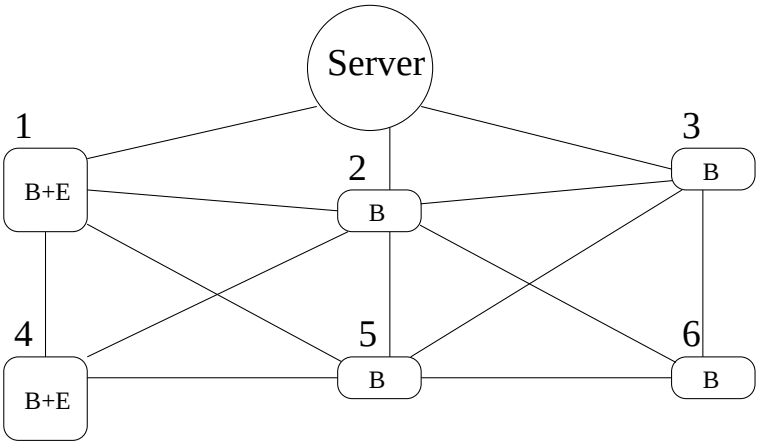

Figure 3.2: With Bandwidth First Scheme (BFS), the base layer (denoted by B) is streamed to all users, and the enhancement layer (denoted by E) is streamed to only users with a large window (i.e., users 1 and 4).

3.2 Dynamic Window Resizing Schemes

Most commercial P2P streaming systems, such as PPStream [60], use single-layer coding because of its simplicity and high coding efficiency. With single-layer coding, the streaming rate to every user is the same and independent of its window size. Therefore, in case of dynamic window resizing, a P2P streaming using single-layer coding can provide zero resizing delay (i.e., design goal 2) and good streaming quality (i.e., design goal 3), however, it is inefficient in bandwidth consumption (i.e., design goal 1). These conclusions are verified by the measurement study of the picture-in-picture feature of PPStream conducted in June 2008 [87].

3.2.1 A Dynamic Window Resizing Scheme based on Adaptive Streaming Alone

A straightforward solution for dynamic window resizing, called Bandwidth First Scheme (BFS), is to use adaptive streaming alone. Figure 3.2 shows a snapshot

of a P2P live streaming system with a mesh overlay, where users 1 and 4 watch the channel in a large window, and users 2, 3, 5, and 6 watch the same channel in a small window. With BFS, the channel is encoded into two layers: a base layer and an enhancement layer. Users with a large window (i.e., users 1 and 4) require both the base layer and the enhancement layer, and users with a small window (i.e., users 2, 3, 5, and 6) require only the base layer.

Even though BFS can effectively reduce the total bandwidth consumption, i.e. design goal 1, it cannot achieve design goals 2 and 3 as illustrated in the following cases.

- Case 1 (cannot achieve design goal 2): Consider the case where user 6 increases its window size from small to large, because he/she becomes interested in the channel and wants a high resolution of the video. Since all neighbors of user 6 have a small window, user 6 must look for new large-window neighbors in order to obtain the enhancement layer. However, the delay to find a new large-window neighbor may be very large, and thus the time it takes for the user to see a high-resolution video is too long to be acceptable.

- Case 2 (cannot achieve design goal 3): Consider user 4 which has a large window and obtains the enhancement layer from user 1. If at some time, user 1 reduces its window size from large to small, and then it does not have the enhancement layer anymore. Since all neighbors of user 4 have a small window, user 4 cannot obtain the enhancement layer from any of them. Consequently, it cannot provide good streaming quality until it establishes a connection to a large-window user or the streaming server as its new neighbor.

3.2.2 Dynamic Window Resizing Schemes based on both Adaptive Streaming and User Cooperation

We propose cooperation among large-window users and small-window users. Intuitively, small-window users contribute part of their bandwidth to forward the enhancement layer so that a large-window user can easily locate and obtain the required enhancement layer not only from large-window users but also from small-window users. There are two possible reasons that a small-window user should forward the enhancement layer.

- First, if a small-window user helps forward the enhancement layer, then it can quickly obtain a high resolution video after resizing to a large window.
- Second, we can also introduce incentive mechanisms so that if a small-window user helps forward the enhancement layer, as a reward it may have a higher priority to obtain the enhancement layer from its neighbors when it changes to a large window. However, these incentive mechanisms are out of the scope of this chapter, and we only consider user cooperation in this chapter.

Problem Formulation

We consider a P2P streaming system with a streaming server O and a set of users V. The set $V = V_L(t) \cup V_S(t)$ consists of two types of users: $V_L(t)$ is the set of users with a large window at time t, and $V_S(t)$ is the set of users with a small window at time t. We assume that the upload capacity of users is the only bottleneck for a P2P live streaming system. That is, the download capacity of a user is higher than the streaming rate, and bandwidth bottlenecks are located at the edges instead of the core of the Internet, which are reasonable assumptions [78] for the current Internet.

When we use adaptive streaming, server O generates a video stream at constant

rate R including the base layer and enhancement layer with streaming rates at R_B and R_E, respectively. A small-window user $i \in V_S(t)$ requires only the base layer, and its required streaming rate is then R_B. A large-window user $i \in V_L(t)$ requires both layers, and its required streaming rate is then $R_B + R_E$.

When we use user cooperation, small-window users contribute part of their bandwidth to forward the enhancement layer so that a large-window user can easily locate and obtain the requirement enhancement layer data not only from large-window users but also from small-window users.

- There are two types of small-window users: some small-window users, called *helpers*, contribute their bandwidth to forward the enhancement layer, and the remaining small-window users, called *non-helpers*, do not forward any part of the enhancement layer. Let $V_H(t)$ denote the set of helpers at time t, and $V_N(t)$ denote the set of non-helpers at time t. We have $V_S(t) = V_H(t) \cup V_N(t)$. Note that, $\emptyset \subseteq V_H(t) \subseteq V_S(t)$. That is, $V_H(t)$ could be empty or could be $V_S(t)$.

- Let R_H denote the download rate of a helper for the enhancement layer. Note that $0 < R_H \leq R_E$. When $R_H < R_E$, a helper only helps forward part of the enhancement layer, and when $R_H = R_E$, a helper helps forward the whole enhancement layer. Note that the upload rate of a helper for the enhancement layer is not limited to R_H, since it can upload R_H multiple times to different neighbors. In order to simplify the protocol design, we assume that every helper has the same R_H.

The streaming quality of a P2P streaming system depends on many factors, such as the overlay topology, the block scheduling algorithm, and the dynamic window resizing scheme. In this chapter, in order to focus on the impact of a dynamic window resizing scheme, we assume that a P2P streaming system has an overlay topology and

a block scheduling algorithm which together can efficiently use the upload capacity of each user. Specifically, we consider a mesh overlay topology, since it is the most popular one used in commercial P2P streaming systems [40, 38, 84]. We consider a random block scheduling algorithm, since it is simple to analyze and it can achieve reasonably good streaming quality [33]. It is worth noting that our proposed schemes can be used together with any block scheduling algorithm [33, 95, 48, 75].

Now we are ready to formally define the dynamic window resizing problem with both adaptive streaming and user cooperation. We summarize the notation used in the chapter in Table 3.1.

Design Goal 1: Efficient Bandwidth Consumption. The total download rate of all users is given as follows, where the first term is the total required streaming rate of all large-window users, the second term is the total required streaming rate of all small-window users, and the last term is the total download rate of all helpers for the enhancement layer.

$$|V_L(t)|(R_B + R_E) + |V_S(t)|R_B + |V_H(t)|R_H \tag{3.1}$$

Since the first two terms are fixed for any dynamic window resizing scheme, we only need to minimize the third term (i.e., $|V_H(t)|R_H$) in order to achieve the first design goal.

Design Goal 2: Good Streaming Quality at Resizing Events (i.e., Short Resizing Delay for Small-Window Users). When a small-window user changes its window size from small to large, the resizing delay depends on whether it can quickly obtain the enhancement layer data from its neighbors. If its neighbors together have sufficient available bandwidth and have the data for the whole enhancement layer, the small-window user can quickly resize to a large window. Otherwise, the small-window

Notation	Description
V	set of users. $V = V_L(t) \cup V_S(t)$
$V_L(t)$	set of large-window users at time t
$V_S(t)$	set of small-window users at time t. $V_S(t) = V_H(t) \cup V_N(t)$
$V_H(t)$	set of helpers at time t
$V_N(t)$	set of non-helpers at time t
$N_i(t)$	set of neighbors of user i. $N_i(t) = N_{Li}(t) \cup N_{Si}(t)$
$N_{Li}(t)$	set of large-window neighbors of user i at time t
$N_{Si}(t)$	set of small-window neighbors of user i at time t. $N_{Si}(t) = N_{Hi}(t) \cup N_{Ni}(t)$
$N_{Hi}(t)$	set of helper neighbors of user i at time t
$D_{Hi}(t)$	number of helper neighbors of user i which have distinct parts of the enhancement layer
$N_{Ni}(t)$	set of non-helper neighbors of user i at time t
P_S	fraction of simultaneous resizing small-window users
R_B	rate of the base layer
R_E	rate of the enhancement layer
R_H	download rate of a helper for the enhancement layer
U	total upload capacity of the system
$u_i(t)$	residual upload capacity of user i at time t
X	PFS parameter: # of enhancement layer substreams
$\tilde{P}_S$	PPS parameter: estimate of P_S

Table 3.1: Notation used in dynamic window resizing

user must find new neighbors with more available bandwidth or more data for the enhancement layer, which however causes longer resizing delay. Therefore, we consider two types of bottlenecks: bandwidth bottleneck and content bottleneck.

- Bandwidth bottleneck: When a small-window user resizes to a large window, it needs to get the whole enhancement layer from its neighbors. If its neighbors together do not have sufficient available bandwidth for the enhancement layer, the small-window user must find new neighbors with more available bandwidth. This is called the bandwidth bottleneck.

  To resolve the bandwidth bottleneck, a non-helper small-window user $i \in V_N(t)$ at time t requires a total of R_E available bandwidth for the whole enhancement layer from all its large-window neighbors (denoted by $N_{Li}(t)$) and helper small-window neighbors (denoted by $N_{Hi}(t)$). Therefore, we require:

$$\sum_{j \in N_{Li}(t) \cup N_{Hi}(t)} \frac{u_j(t)}{\max(1, |N_{Sj}(t)|P_S)} \geq R_E \quad \forall i \in V_N(t) \tag{3.2}$$

  where $u_j(t)$ denotes the residual upload bandwidth of user j at time t which is the difference between its upload capacity and its average upload rate measured in a recent time period before time t. The upload capacity of a user can be measured [78] when the user joins the system and periodically after that, and the average upload rate can be measured by keeping track of the number of packets uploaded in a recent time period. We assume that the fraction of small-window users who simultaneously resize to a large window at a time is fixed, and let P_S denote this fixed fraction. Note that our proposed solutions do not make this assumption, and work with dynamic P_S. To understand the fraction $\frac{u_j(t)}{\max(1,|N_{Nj}(t)|P_S)}$, let's consider a neighbor j of user i. User j has $|N_{Sj}(t)|$ small-window neighbors, among which $|N_{Sj}(t)|P_S$ neighbors will resize to a large

window. Therefore, the residual upload bandwidth $u_j(t)$ of user j will be shared by $|N_{Sj}(t)|P_S$ resizing neighbors.

To resolve the bandwidth bottleneck, a helper small-window user $i \in V_H(t)$ at time t requires a total of $R_E - R_H$ available bandwidth for the remaining enhancement layer from all its large-window neighbors and helper small-window neighbors. Therefore, we require:

$$\sum_{j \in N_{Li}(t) \cup N_{Hi}(t)} \frac{u_j(t)}{\max(1, |N_{Sj}(t)|P_S)} \geq R_E - R_H \quad \forall i \in V_H(t) \tag{3.3}$$

- Content bottleneck: When a small-window user resizes to a large window, it needs to get the whole enhancement layer from its neighbors. If its neighbors together do not have the whole enhancement layer, the small-window user must find new neighbors with more enhancement layer data. This is called the content bottleneck.

  To resolve the content bottleneck, a small-window user $i \in V_S(t)$ at time t requires:

  $$|N_{Li}(t)|R_E + D_{Hi}(t)R_H \geq R_E \quad \forall i \in V_S(t) \tag{3.4}$$

  where $D_{Hi}(t)$ denote the number of helper neighbors which have distinct parts of the enhancement layer. For example, if two helpers have the same part of the enhancement layer, they are counted as only one helper in $D_{Hi}(t)$. Intuitively, this constraint means that a small-window user i must have at least one large-window neighbor (or the streaming server), or its helper neighbors together have the whole enhancement layer data.

Design Goal 3: Good Streaming Quality at Other Times. If a dynamic window resizing scheme can achieve design goal 2, then it, together with an efficient overlay

topology and an efficient block scheduling algorithm, can also achieve design goal 3.

When Inequalities (3.2), (3.3) and (3.4) hold, we can construct a P2P network which achieves good streaming quality. Thus, the *goal of dynamic window resizing problem* with both adaptive streaming and user cooperation is

$$\min |V_H(t)|R_H \quad \text{subject to constraints (3.2), (3.3) and (3.4)} \tag{3.5}$$

Proposed Solutions

In order to design simple and efficient P2P streaming systems, we follow two approaches to tackle the dynamic window resizing problem (3.5): keep $|V_H(t)|$ fixed and then minimize R_H, or keep R_H fixed and then minimize $|V_H(t)|$. Therefore, we have the following two schemes.

- First scheme: Every small-window user is a helper. That is, $V_H(t) = V_S(t)$. Every helper forwards only a fraction of the enhancement layer. That is, $R_H < R_E$. In this case, we should minimize R_H in order to minimize $|V_H(t)|R_H$. Partial Forwarding Scheme (PFS) [53] described in Section 3.3 follows this approach.

- Second scheme: Only some small-window users are helpers. That is, $V_H(t) \subseteq V_S(t)$. Every helper forwards the whole enhancement layer. That is, $R_H = R_E$. In this case, we should minimize $|V_H(t)|$ in order to minimize $|V_H(t)|R_H$. Partial Participation Scheme (PPS) [88] described in Section 3.4 follows this approach.

In this chapter, we compare the performance of PFS and PPS with two other reference schemes.

- Bandwidth-First Scheme (BFS) described in Section 3.2.1 does not have any helpers. That is $|V_H(t)| = 0$, and then $|V_H(t)|R_H = 0$. Therefore, BFS consumes

the minimum amount of bandwidth. However, as we discussed before, BFS may have long resizing delay and poor streaming quality.

- Quality-First Scheme (QFS): Every small-window user is a helper, and every helper forwards the whole enhancement layer. That is, $V_H(t) = V_S(t)$ and $R_H = R_E$. Therefore, QFS can achieve the smallest resizing delay and the best streaming quality. However, QFS consumes the maximum amount of bandwidth.

Intuitively, if we consider both variables $|V_H|$ and R_H together (referred to as joint minimization), we may obtain a better solution in the sense that we can further reduce the total amount of streaming traffic. There are two major reasons that we choose to separately minimize each variable (referred to as separate minimization).

- 1) Joint minimization implies that only some small-window users are helpers, and each helper has only part (possibly different fractions) of the enhancement layer. Even though joint minimization can further reduce the total amount of streaming traffic, it considerably complicates the design and implementation of the user cooperation scheme and also requires more control messages.

- 2) Our simulation results shown in Section 3.5 demonstrate that separate minimization consumes significantly less bandwidth than QFS (i.e., the upper bound), and consumes only slightly more bandwidth than BFS (i.e., the lower bound). Therefore, we feel that even though joint minimization can further reduce the total amount of streaming traffic, it can consume only slightly less bandwidth than separate minimization.

3.3 Partial Forwarding Scheme (PFS)

In this section, we explain how PFS [53] works and how to set its parameters. The evaluation of PFS is presented in Section 3.5.

3.3.1 General Idea

In PFS, the enhancement layer is divided into $X \geq 1$ substreams, where X is the only parameter of PFS. Every small-window user is a helper and helps to forward a substream of the enhancement layer. For example, in Figure 3.3, we have $V_L(t) = \{1, 4\}$, and $V_S(t) = V_H(t) = \{2, 3, 5, 6\}$. If at some time small-window user 6 changes to a large window, it can quickly get substream 2 from helpers 3 and 5, and thus achieve very short window resizing delay. If at some time large-window user 1 changes to a small window, it has only the base layer and one substream, say substream 2. In this case, its large-window neighbor 4 can still get substream 1 from helper 2 and get substream 2 from helpers 1 and 5, and thus it can still maintain good streaming quality.

Since every small-window user is a helper, we have $V_H(t) = V_S(t)$. Since a helper only needs to download one substream of the enhancement layer, we have $R_H = R_E/X$. Therefore, we should minimize $|V_H(t)|R_H = |V_S(t)|R_E/X$ in order to solve dynamic window resizing problem (3.5). Since both $|V_S(t)|$ and R_E are independent of PFS, the *goal of PFS* is to

$$\max X \quad \text{subject to constraints (3.2), (3.3) and (3.4)} \tag{3.6}$$

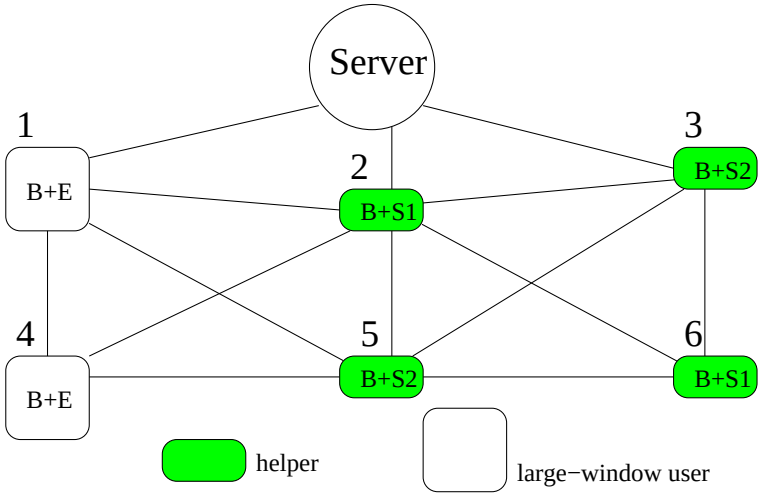

Figure 3.3: With PFS, *every* small-window user is a helper, and helps to forward a *substream* of the enhancement layer. In this figure, the whole enhancement layer (denoted by E) is divided into two substreams (denoted by $S1$ and $S2$).

3.3.2 Implementation Details

There are various ways [43] to divide the enhancement layer to substreams. A simple way is to divide the enhancement layer using its packet sequence number, as illustrated in Figure 3.4. A small-window user is randomly assigned to one substream, and then it keeps forwarding this substream until it leaves the system. In addition, every user should monitor the status of its neighbors, and maintain a sufficient (as typically required by the overlay topology and block scheduling algorithm) and balanced (i.e. with both large-window neighbors and small-window neighbors, and with small-window users with different substreams) group of neighbors. Peers can get the status information by periodically exchanging the local information. This information can be added in the ordinary video packets.

Ideally, all small-window peers in PFS are required to be helpers. But in practice, different users have different amounts of upload bandwidth. Some users may have limited upload capacities which are less than $R_B + R_E$, and they can still work as

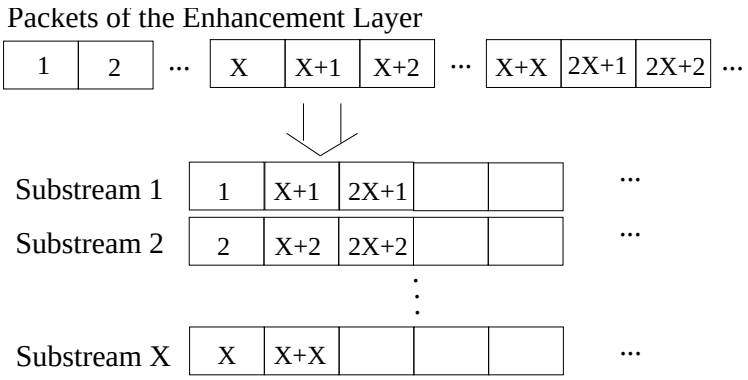

Figure 3.4: A simple way to divide the enhancement layer into X substreams.

helpers as long as their upload capacities are at least $R_B + R_E/X$. Some user may have very limited upload capacities which are even lower than $R_B + R_E/X$, and these peers are not required to play as helpers.

3.3.3 Parameter Setting

In this section, we study the impact of parameter X on the bandwidth bottleneck constraints (3.2) and (3.3) and on content bottleneck constraint (3.4).

Impact of X on Bandwidth Bottleneck Constraints

With PFS, every small-window user is a helper, so we do not need to consider bandwidth bottleneck constraint (3.2) for non-helpers. With PFS, we have $V_H(t) = V_S(t)$ and $R_H = R_E/X$, and therefore bandwidth bottleneck constraint (3.3) for a small-window user i (i.e., a helper) can be rewritten as follows.

$$\sum_{j \in N_{Li}(t) \cup N_{Hi}(t)} \frac{u_j(t)}{\max(1, |N_{Sj}(t)|P_S)} \geq R_E \left(1 - \frac{1}{X}\right) \quad \forall i \in V_S(t) \tag{3.7}$$

Note that, each small-window user requires a total of R_E/X enhancement layer data from its neighbors, and thus the residual upload capacity (i.e., $u_j(t)$) of its neighbors

depends on X. Since both sides of Inequality (3.7) depend on X, it is hard to see the impact of X on the inequality. Therefore, below we consider the worst case scenario for bandwidth bottleneck constraint (3.3).

Let's consider a case where every user is a small-window user (i.e., $N_{Li}(t) = \emptyset\ \forall i$, $V_L(t) = \emptyset$, and thus $N_{Hi}(t) = N_{Si}(t) = N_i(t)\ \forall i$ and $V_H(t) = V_S(t) = V(t)$) and every user switches to a large window at that time (i.e., $P_S = 1$). This is the worst case, since the case requires the largest amount of residual upload capacity. In this case, bandwidth bottleneck constraint (3.3) for a user i can be rewritten as follows.

$$\sum_{j \in N_i(t)} \frac{u_j(t)}{|N_j(t)|} \geq R_E \left(1 - \frac{1}{X}\right) \qquad \forall i \in V(t) \tag{3.8}$$

We sum the above inequality over all users, and we have

$$\sum_{i \in V(t)} \sum_{j \in N_i(t)} \frac{u_j(t)}{|N_j(t)|} \geq |V(t)| R_E \left(1 - \frac{1}{X}\right) \tag{3.9}$$

The left side can be rewritten as follows, where U denotes the total upload capacity of the P2P streaming system.

$$\sum_{i \in V(t)} \sum_{j \in N_i(t)} \frac{u_j(t)}{|N_j(t)|} = \sum_{j \in V(t)} \sum_{i \in N_j(t)} \frac{u_j(t)}{|N_j(t)|} = \sum_{j \in V(t)} u_j(t) = U - |V(t)|(R_B + \frac{R_E}{X}) \tag{3.10}$$

We can see that Inequality (3.9) holds as long as $U \geq |V(t)|(R_B + R_E)$. That is, the whole P2P streaming system should have sufficient upload capacity to stream both the base layer and the enhancement layer to every user. Obviously, this condition must be true for the worst case scenario for any P2P streaming scheme supporting dynamic window resizing including PFS and PPS; otherwise, the system simply does not have sufficient upload capacity when every user has a large window. If $U \geq |V(t)|(R_B + R_E)$ and the system upload capacity is fairly allocated to every user, which is true for most

P2P streaming systems [40], Inequality (3.8) also holds.

Finally, as long as the whole P2P streaming system has sufficient upload capacity (i.e., $U \geq |V(t)|(R_B+R_E)$), and the system upload capacity is fairly allocated to every user, bandwidth bottleneck constraint (3.3) is satisfied. Note that, parameter X does not affect the constraint, and intuitively this is because PFS uniformly distributes the extra traffic load to every small-window user.

Impact of X on Content Bottleneck Constraint

The content bottleneck constraint (3.4) for a small-window user i can be rewritten as

$$|N_{Li}(t)|R_E + \frac{D_{Hi}(t)}{X}R_E \geq R_E \quad \forall i \in V_S(t) \tag{3.11}$$

Therefore, in order to satisfy this constraint, we should choose at least one large-window neighbor for a small-window user (i.e., $|N_{Li}(t)| \geq 1$), or make sure that all small-window neighbors together have all X distinct enhancement layer substreams (i.e., $D_{Hi}(t) = X$). If there are very few large-window users in the system, it is impossible to have a large-window neighbor for each small-window user. Therefore, below we study how to choose the value of X so that all small-window neighbors of a user together have all X distinct substreams with a high probability.

Let's consider a user i with a total of $N_i(t)$ neighbors at time t. If user i does not have any large-window user, all $N_i(t)$ neighbors are small-window neighbors. That is, $N_i(t) = N_{Si}(t) = N_{Hi}(t)$. Each small-window user is randomly assigned with a substream, and therefore, the probability that the $N_i(t)$ neighbors of user i together have all X distinct substreams can be calculated by $P\left(N_i(t), X\right)$ as follows.

$$P\left(N_i(t), X\right) = \sum_{k=0}^{X-1}(-1)^k\binom{X}{k}\left(1-\frac{k}{X}\right)^{N_i(t)} \tag{3.12}$$

Intuitively, expression $\binom{X}{k}\left(1-\frac{k}{X}\right)^{N_i(t)}$ is the probability that at least k substreams are unavailable among the $N_i(t)$ neighbors, and expression $(-1)^k$ is introduced according to the inclusion-exclusion principle [15].

As discussed before, to achieve the PFS goal (3.6), we should choose an X as large as possible. To satisfy the content bottleneck constraint (3.4), we should choose the value of X so that $P\left(N_i(t), X\right)$ is close to 1. However, probability $P\left(N_i(t), X\right)$ is a decreasing function of X. That is, there is a tradeoff between the PFS goal and the content bottleneck. Therefore, we choose the maximum X such that probability $P\left(N_i(t), X\right)$ is at least 90%. Figure 3.6 plots the maximum X so that $P\left(N_i(t), X\right) \geq 90\%$ as $N_i(t)$ varies from 1 to 100. Figure 3.7 plots the ratio of the maximum X and $N_i(t)$. For example, for a P2P streaming system with 32 neighbors per user, the ratio is about 0.25. That is, the maximum X so that $P\left(32, X\right) \geq 90\%$ is about $32 \times 0.25 = 8$. As another example, for a system with 64 neighbors per user, the ratio is about 0.2. That is, the maximum X so that $P\left(64, X\right) \geq 90\%$ is about $64 \times 0.2 = 13$.

Finally, if the average number of neighbors (denoted by N) of a P2P system is between 10 and 50, we recommend to set X to $N \times 0.25 = N/4$. If N is between 50 and 100, we recommend to set X to $N \times 0.2 = N/5$.

Extreme Cases of X

We can see that X is a number in $[1, +\infty)$. There are two extreme cases.

- When X approaches $+\infty$, a small-window user does not forward any enhancement layer packet. This extreme case is Bandwidth-First Scheme (BFS).
- When X is 1, every user has both the base layer and the whole enhancement layer. This extreme case is Quality-First Scheme (QFS).

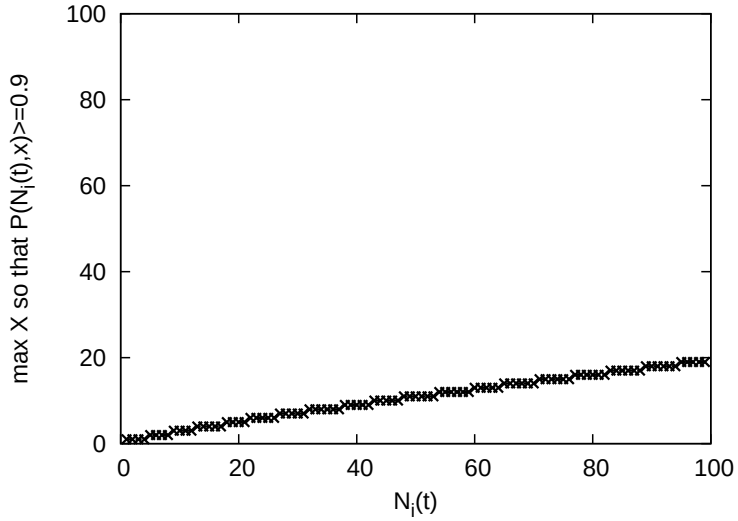

Figure 3.5: The maximum X such that probability $P(N_i(t), X)$ is at least 90%, as $N_i(t)$ increases from 1 to 100.

Compared to BFS, a small-window user in PFS experiences shorter window resizing delay when it changes to a large window for the following reasons: (1) It already has one substream of the enhancement layer and it only needs to find the other $X-1$ substreams instead of the whole enhancement layer; (2) It can get the enhancement layer not only from its large-window neighbors, but also from its small-window neighbors. Compared to BFS, a large-window user can also benefit from PFS. If one large-window neighbor of a large-window user changes to a small window, this user can still get the enhancement layer not only from other large-window neighbors, but also from its small-window neighbors.

Compared to QFS, PFS is more bandwidth-efficient. A small-window user in PFS requests only one substream instead of the whole enhancement layer. The larger the X is, the less the amount of bandwidth consumed by PFS.

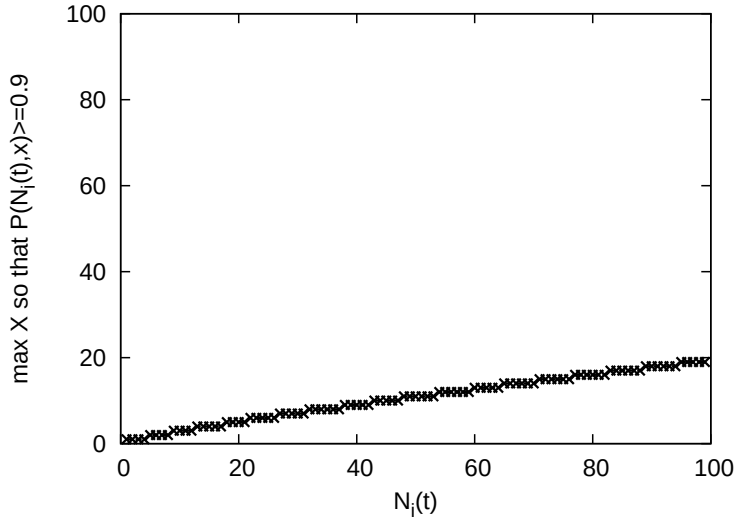

Figure 3.6: The maximum X such that probability $P(N_i(t), X)$ is at least 90%, as $N_i(t)$ increases from 1 to 100.

3.4 Partial Participation Scheme (PPS)

In this section, we propose Partial Participation Scheme (PPS), which is an extension of our previous work [88]. The evaluation of PPS is presented in Section 3.5.

3.4.1 General Idea

In PPS, only some small-window users are helpers. That is, $V_H(t) \subseteq V_S(t)$. But every helper helps to forward the whole enhancement layer. That is, $R_H = R_E$. With PPS, a large-window user can find and obtain the enhancement layer not only from other large-window users, but also from these helpers. For example, in Figure 3.8, we have $V_L(t) = \{1, 4\}$, and $V_S(t) = V_H(t) \cup V_N(t)$, where $V_H(t) = \{2\}$ and $V_N(t) = \{3, 5, 6\}$. When small-window user 6 changes to a large window, it can quickly get the enhancement layer from helper 2. If large-window user 1 changes to a small window, its large-window neighbor 4 can still get the enhancement layer from helper 2.

Since a helper needs to download the whole enhancement layer, we have $R_H = R_E$.

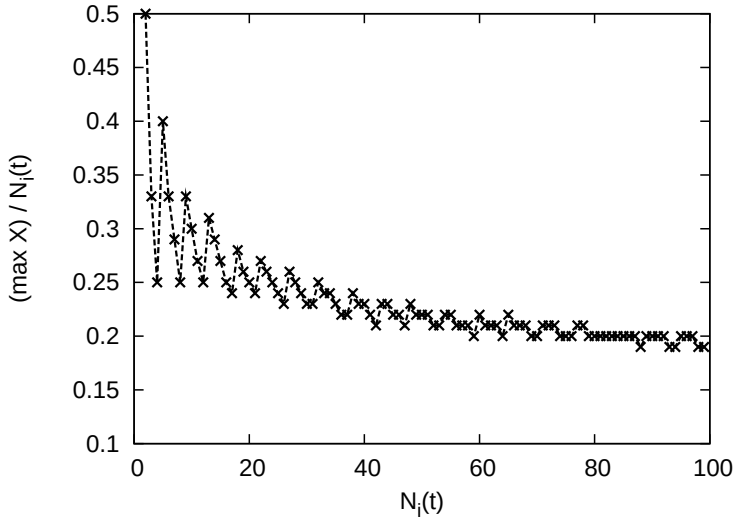

Figure 3.7: If $10 \leq N_i(t) < 50$, the ratio is around 0.25. If $50 \leq N_i(t) \leq 100$, the ratio is around 0.20.

Therefore, we should minimize $|V_H(t)|R_H = |V_H(t)|R_E$ in order to solve dynamic window resizing problem (3.5). Since R_E is independent of PPS, the *goal of PPS* is to

$$\min |V_H(t)| \text{ subject to constraints (3.2), (3.3) and (3.4)} \tag{3.13}$$

We have the following design guidelines in order to minimize $|V_H(t)|$ while still satisfying constraints (3.2) and (3.4). Note that, since $R_H = R_E$, a helper can resize to a large window without any resizing delay. Therefore, we do not need to consider bandwidth bottleneck constraint (3.3) for helpers.

- *Design Guideline 1:* PPS should select small-window users with high upload capacity as helpers.

This is to minimize the total number of helpers (i.e., $|V_H(t)|$) while maximizing the total available bandwidth of all helpers (i.e., maximize the $u_j(t)$ in bandwidth bottleneck constraint (3.2) for non-helpers). In addition, a user with higher upload capacity tends to have more number of neighbors (peers with higher upload capacity can serve

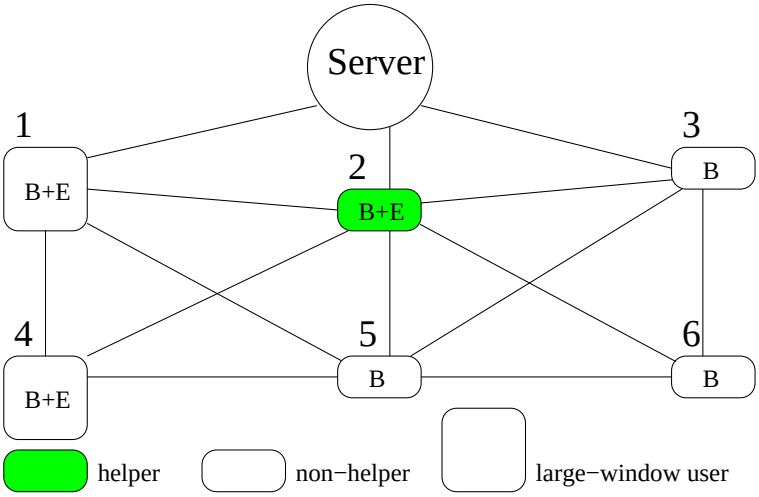

Figure 3.8: With PPS, *some* small-window users, called helpers, help to forward the *whole* enhancement layer.

more requests, and their neighbors are willing to connect with them). Therefore, a helper with higher upload capacity can also benefit more number of large-window users.

In order to satisfy the bandwidth bottleneck constraint (3.2) for non-helpers, we also have the following design guideline.

- *Design Guideline 2:* PPS should maintain an appropriate number of helpers in the system, so that when a small-window user resizes to a large window, its current large-window neighbors and helper neighbors together have sufficient residual upload bandwidth to forward the whole enhancement layer to the user.

Intuitively, the second design guideline adjusts the number of helpers (i.e., $|V_H(t)|$) in the system to satisfy constraint (3.2).

Finally, in order to satisfy the content bottleneck constraint (3.4), PPS should maintain at least one larger-window user (i.e., $|N_{Li}(t)| \geq 1$) or one helper (i.e., $|N_{Hi}(t)| \geq 1$ and thus $D_{Hi}(t) = 1$) as a neighbor of a user. Note that, this is naturally true if PPS follows the second design guideline.

3.4.2 Implementation Details

All users in PPS form a mesh-based overlay. A user may have different types of neighbors. For example, a large-window user may have helper neighbors and non-helper neighbors, in addition to large-window neighbors. When a large-window user and a helper are neighbors, they can forward both the base layer and the enhancement layer to each other. When a helper resizes to a large window, it can resize to a large window without any resizing delay.

Every user periodically exchanges information, including the buffer map, user state, neighbor information, and residual bandwidth, with its neighbors. A buffer map of user i indicates which blocks it has already obtained. The user state of user i indicates whether it is a large-window user, a helper, or a non-helper. The neighbor information of user i contains the number of its large-window neighbors (i.e., $|N_{Li}|(t)$), the number of its helper neighbors (i.e., $|N_{Hi}(t)|$), and the number of its non-helper neighbors (i.e., $|N_{Ni}(t)|$). The residual bandwidth of user i (i.e., $u_i(t)$) is the difference between its upload capacity and its average upload rate measured in a recent time period. The total amount of control traffic is comparable to other P2P streaming schemes, since the buffer map is common for P2P streaming schemes, and only a few more bytes are needed to represent the user state, neighbor information, and residual bandwidth.

A large-window user or a helper maintains a list of good neighbors by periodically finding new neighbors with high residual bandwidth and removing the current neighbors with a low upload rate. Below, we describe how a non-helper maintains its neighbors and how a new user joins the system by following the two design guidelines.

Neighbor Maintenance

Each non-helper periodically monitors the status of its neighbors by exchanging user information with its neighbors, and then maintains a list of good neighbors by following the two design guidelines. Specifically, non-helper i gets information $u_j(t)$ and $|N_{Nj}(t)|$ from each neighbor j, and then it periodically checks the bandwidth bottleneck constraint (3.2) by checking whether the following condition is satisfied.

$$\sum_{j \in N_{Li}(t) \cup N_{Hi}(t)} \frac{u_j(t)}{\max(1, |N_{Nj}(t)|\tilde{P}_S)} \geq R_E \tag{3.14}$$

where $\tilde{P}_S$ is a PPS parameter used to estimate the value of P_S which is the fraction of small-window users who simultaneously resize to a large window at a time. This follows the second design guideline.

Every time non-helper i finds that Inequality (3.14) is not satisfied, it takes the following two steps.

- *Step 1: find one more large-window neighbor or helper neighbor.* It contacts its neighbors to get the list of their neighbors. Among the users in this list, non-helper i finds the one, say user j, which has the highest residual upload bandwidth and is either a large-window user or a helper, and then establishes a connection with user j. If the total number of neighbors of non-helper i becomes larger than a threshold, the neighbor with the lowest streaming rate to non-helper i will be disconnected. If Inequality (3.14) is still not satisfied after adding new neighbor j, then non-helper i takes the second step.

- *Step 2: submit a request for one more helper.* Among all non-helper neighbors, non-helper i finds the one, say user j, with the highest residual upload bandwidth, and then sends a request which asks user j to change to a helper. This

follows the first design guideline.

Every time non-helper i finds that Inequality (3.14) is satisfied, it takes the following two steps.

- *Step 1: determine whether there are too many helpers.* It checks whether the left-hand side Inequality (3.14) is considerably greater than (e.g., more than 1.5 times) the right-hand side, and checks whether the total number of its large-window neighbors and helper neighbors is more than half of the total number of its neighbors. If so, it takes the second step.

- *Step 2: submit a request to reduce the number of helpers.* Among all helper neighbors, non-helper i finds the one, say user j, which has the lowest upload bandwidth, and then sends a request which asks user j to change back to a non-helper.

When a peer receives requests for more helpers or for reducing helpers, it makes the decision based on its local information. In our simulation, a non-helper peer becomes a helper when it receives the request for one more helper, while it terminates the enhancement layer forwarding after receiving 3 consecutive reducing requests.

New Users

When a new user joins the system, it should obtain a list of neighbors satisfying the following conditions.

- First, if the new user has a small window, all of its neighbors together should be able to provide the base layer to it. If the new user has a large window, all of its neighbors together should be able to provide both the base layer and the

enhancement layer to it. For a small-window new user i, we have

$$\sum_{j \in N_i(t)} u_j(t) \geq R_B \tag{3.15}$$

For a large-window new user i, we have

$$\sum_{j \in N_i(t)} u_j(t) \geq R_B + R_E \tag{3.16}$$

- Second, if the new user has a small window, it is a non-helper and all its large-window neighbors and helper neighbors should satisfy Inequality (3.14) according to the second design guideline.

In order to satisfy the above two conditions, a new user first contacts the track server to get a random list of users. If these users cannot satisfy the first condition (i.e., Inequality (3.15) or Inequality (3.16)), the new user contacts the neighbors of its current neighbors until finally the first condition is satisfied. If the new user has a small window and the second condition is not satisfied (i.e., Inequality (3.14)), the new user follows the neighbor maintenance steps described in the previous subsection.

3.4.3 Parameter Setting

PPS parameter $\tilde{P}_S$ should be set according to the fraction of small-window users who simultaneously resize to a large window at a time. In practice, this fraction may change over time, and therefore, we set $\tilde{P}_S$ to the average fraction of the system measured in a recent time period. Note that, a system with a large number of resizing events does not necessarily have many synchronized resizing events, since these resizing events may happen at different times.

The setting of $\tilde{P}_S$ is also influenced by the bandwidth distribution of user. In-

tuitively, the greater the number of high bandwidth users, the greater the number of feasible values of $\tilde{P}_S$. In a system with many high-bandwidth users, we can set very high $\tilde{P}_S$ without impairing the system streaming quality. However, in a system with few high-bandwidth users, a very high $\tilde{P}_S$ causes many small-window users to download the extra enhancement layer, which in turn aggravates the competition for the limited bandwidth. Therefore, in a system with few high-bandwidth users, an inappropriately high $\tilde{P}_S$ may impair the system streaming quality. In general, it is safer for PPS to have a small $\tilde{P}_S$ instead of a large one.

There are two extreme cases of PPS.

- When $\tilde{P}_S$ is very small, like 1%, denominator $\max(1, |N_{Nj}(t)|\tilde{P}_S)$ in Inequality (3.14) becomes 1. In this case, PPS assumes that among all non-helper neighbors of user j, only one of them may change to a large window in the near future. Therefore, there are very few helpers. That is, $|V_H(t)|$ is very small. Note that, this extreme case consumes slightly more bandwidth than BFS.

- When $\tilde{P}_S$ is 100%, denominator $\max(1, |N_{Nj}(t)|\tilde{P}_S)$ in Inequality (3.14) becomes $|N_{Nj}(t)|$. In this case, PPS assumes that all non-helper neighbors of user j may simultaneously change to a large window in the near future. Therefore, this extreme case has more helpers than the first extreme case. Typically, users in a P2P streaming system have heterogeneous upload capacity, and the upload capacity of some users (e.g., 1Mbps) is more than ten times of that of some other users (e.g., 128Kbps). Since PPS follows the first design guideline and select users with high upload capacity as helpers, the total number of helpers in this extreme case is still much less than the total number of small-window users. That is, $|V_H(t)| \ll |V_S(t)|$. Note that this extreme case consumes much less bandwidth than QFS.

3.4.4 Bandwidth Consumption of All Schemes

In this subsection, we analytically compare the total bandwidth consumption of all schemes according to (3.1).

For PFS with parameter X, we have $V_H(t) = V_S(t)$ and $R_H = R_E/X$. Therefore, its total bandwidth consumption is

$$|V_L(t)|(R_B + R_E) + |V_S(t)|R_B + |V_S(t)|\frac{R_E}{X} \tag{3.17}$$

For PPS with parameter $\tilde{P}_S$, we have $V_H(t) \subseteq V_S(t)$ and $R_H = R_E$. Therefore, its total bandwidth consumption is

$$|V_L(t)|(R_B + R_E) + |V_N(t)|R_B + |V_H(t)|R_E \tag{3.18}$$

where both $|V_N(t)|$ and $|V_H(t)|$ depend on its parameter $\tilde{P}_S$ due to Inequality (3.14).

For BFS, we have $V_H(t) = \emptyset$ and $R_H = 0$. Therefore, its total bandwidth consumption is

$$|V_L(t)|(R_B + R_E) + |V_S(t)|R_B \tag{3.19}$$

For QFS, we have $V_H(t) = V_S(t)$ and $R_H = R_E$. Therefore, its total bandwidth consumption is

$$|V_L(t)|(R_B + R_E) + |V_S(t)|(R_B + R_E) \tag{3.20}$$

3.5 Simulation Results

In this section, we evaluate the performance of PFS and PPS using our packet-level P2P streaming simulator, which is an extension of the P2P streaming simulator originally developed by Zhang *et al* [95]. We simulate a P2P streaming system,

where users join and leave the system following a real-world trace of a P2P streaming system [95]. We simulate a mesh overlay topology. We simulate the random block scheduling algorithm [33], but other block scheduling algorithms should not change the relative performance of these dynamic window resizing schemes. There are at most 1000 concurrent users, and each with an average of about 16 neighbors by default. Users have heterogeneous upload capacity, and on average there are 23% of users with 1 Mbps rate, 46% of users with 384 Kbps rate, and 31% of users with 128 Kbps rate. The upload capacity of the streaming server is 2 Mbps. The stream is encoded into the base layer and one enhancement layer at a rate of 100 Kbps and 230 Kbps, respectively.

We simulate the following two types of dynamic window resizing.

- *Random Window Resizing*: Each user independently resizes its window size. A small-window user changes to a large window after an exponentially distributed time with mean $1/\lambda$. A large-window user changes to a small window after an exponentially distributed time with mean $1/\mu$. In this case, the average resizing interval is $(\frac{1}{\lambda}+\frac{1}{\mu})/2 = (\lambda+\mu)/(2\lambda\mu)$. There is an average of $\mu/(\lambda+\mu)$ fraction of small-window users, and an average of $\lambda/(\lambda+\mu)$ fraction of large-window users.

- *Synchronized Window Resizing*: A P_S percent of small-window users simultaneously change to a large window. For example, some users watching a movie may resize their windows when commercial ads are played during the movie, and then simultaneously change back to the full-screen window after the ads.

We simulate the following six dynamic window resizing schemes.

- *PFS with* $X = 16$: In this case, PFS divides the enhancement layer into 16 substreams. By default, every user in our simulation has about 16 neighbors,

so 16 is the upper bound of X.

- *PFS with* $X = 4$: In this case, PFS divides the enhancement layer into 4 substreams. This is our recommended value of X.

- *PPS with* $\tilde{P}_S = 1\%$. In this case, the value of $\max(1, |N_{Nj}(t)|\tilde{P}_S)$ in Inequality (3.14) is 1. That is, PPS assumes that among all non-helper neighbors of a user, only one of them may change to a large window in the near future.

- *PPS with* $\tilde{P}_S = 100\%$. In this case, the value of $\max(1, |N_{Nj}(t)|\tilde{P}_S)$ in Inequality (3.14) is $|N_{Nj}(t)|$. That is, PPS assumes that all non-helper neighbors of a user may simultaneously change to a large window in the near future.

- *BFS*. This is simulated as a reference scheme which consumes the least amount of bandwidth.

- *QFS*. This is simulated as a reference scheme which achieves the shortest resizing delay and best streaming quality.

According to our three design goals, we evaluate the following performance metrics for each scheme and for each simulation.

- *Average Bandwidth Consumption*: This is measured by the ratio of the total download rate of all users to the total number of users. Note that, this includes both the download rate of video packets and also the rate of control packets.

- *Resizing Delay of Small-Window Users*: This is indirectly measured by the average playback continuity between 1.5 and 2 seconds after a small-window user starts to resize to a large window. A high playback continuity implies that the user has short resizing delay.

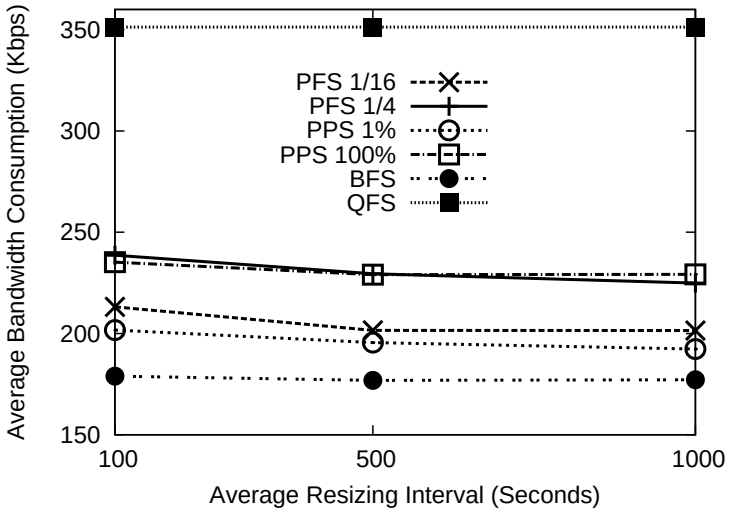

Figure 3.9: Every PFS and PPS scheme's consumed bandwidth is close to BFS, and much less than that of QFS.

- *Average Stream Quality*: This is measured by the average playback continuity of all users in the whole simulation. For small-window peers, the playback continuity is the percentage of base layer packets being received on time. While for a large-window peer, it is the percentage of both base layer and enhancement layer packets received on time.

3.5.1 Impact of the Resizing Interval

In this subsection, we investigate the impact of the resizing interval on the performance of dynamic window resizing schemes. We simulate the random window resizing. We select the values of λ and μ so that there is an average of 20% of large-window users (i.e. $\lambda/(\lambda+\mu) = 20\%$), and the average resizing interval varies from 100 seconds, to 500 seconds, and 1000 seconds (i.e., $(\lambda + \mu)/(2\lambda\mu)$ =100, 500, and 1000).

Figure 3.9 shows the average bandwidth consumption of each scheme. We can see that BFS consumes the least amount of bandwidth, whereas QFS consumes the most

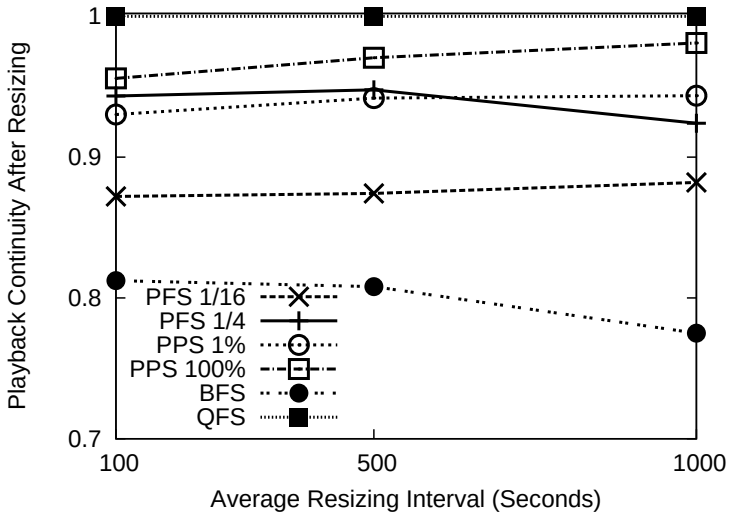

Figure 3.10: PPS with similar bandwidth consumption as PFS can achieve shorter resizing delay than PFS.

amount of bandwidth. The average download rate of BFS includes the download rate of video packets (i.e., $(|V_L(t)|(R_B+R_E)+|V_S(t)|R_B\ )/|V(t)| = 0.2\times 330+0.8\times 100 =$ 146 Kbps) and that of control packets (about 20 Kbps). The average download rate of QFS includes the download rate of video packets (i.e., $(|V_L(t)|(R_B+R_E)+|V_S(t)|(R_B+R_E))/|V(t)| = 330$ Kbps) and that of control packets (about 20 Kbps). We can also see that *the amount of bandwidth consumed by every PFS and PPS scheme is close to that of BFS, and much less than that of QFS.* As we expected, PFS with a smaller X consumes more bandwidth, and PPS with a higher $\tilde{P}_S$ consumes more bandwidth. We notice that PFS with $X = 4$ consumes approximately the same amount of bandwidth as PPS with $\tilde{P}_S = 100\%$.

We can also see the average resizing interval has little impact on the average bandwidth consumption. As the average resizing interval decreases, the average bandwidth consumption increases slightly for the following reason. A user requests and downloads a video packet up to Δt seconds before its playback deadline to make sure that the stream can be played smoothly. In our simulator, Δt is set to 10 seconds. When

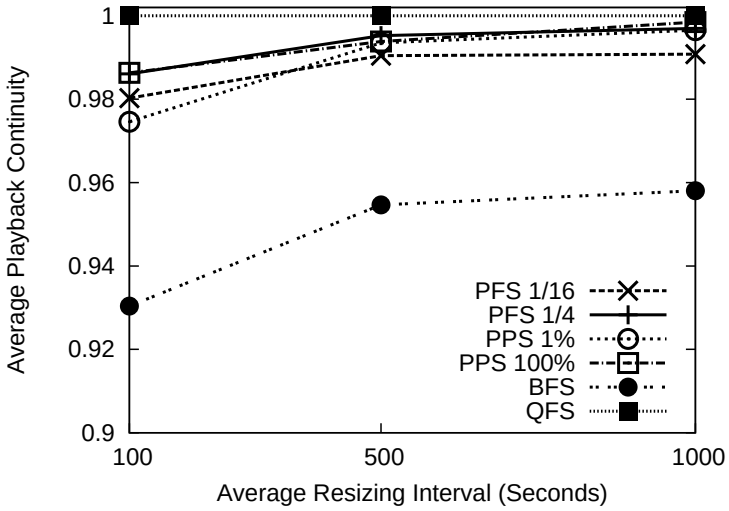

Figure 3.11: All PFS and PPS schemes achieve streaming quality very close to QFS, and much better than BFS.

a large-window user resizes to a small window at time t, it has already downloaded the enhancement layer packets between time t and $t + \Delta t$. Therefore, the smaller the average resizing interval, the higher the impact of this Δt.

Figure 3.10 shows the average playback continuity between 1.5 and 2 seconds after a small-window user starts to resize to a large window. This is used to indirectly show the resizing delay. We can see that BFS has the lowest playback continuity (i.e., the longest resizing delay), and QFS has the highest playback continuity (actually 100%, that is, no resizing delay). As we expected, PFS with a smaller X has higher playback continuity, and PPS with a higher $\tilde{P}_S$ has higher playback continuity. We can also see that PPS with $\tilde{P}_S = 100\%$ has higher playback continuity than PFS with $X = 4$, even though they consume approximately the same amount of bandwidth in Figure 3.9. That is, *PPS can achieve shorter resizing delay than PFS while consuming similar amount of bandwidth as PFS. In other words, PPS can more efficiently use the bandwidth of helpers than PFS.*

We notice that sometimes frequent resizing events may shorten the resizing delay

of BFS and PFS, which seems to be counterintuitive. This is because users in BFS and PFS do not maintain their neighbor lists as actively as users in PPS, and sometimes frequent resizing events may make the large-window users more evenly distributed in the overlay. Therefore, in case of frequent resizing events, users resizing to a large window may have a better chance of obtaining the enhancement layer directly from their neighbors. Consequently, although the overall system streaming quality is usually impaired by frequent resizing events, the local resizing delay of users may be shortened by them.

Figure 3.11 shows the average playback continuity of all users in the whole simulation, as a measurement of the streaming quality. We can see that all schemes achieve better streaming quality as the average resizing interval becomes longer. This is because every resizing event may change the packets that a user needs to download and upload, and may change the neighbors of the user, and therefore interferes the video streaming. We can also see that *all PFS and PPS schemes achieve streaming quality very close to QFS, and much better than BFS.*

3.5.2 Impact of the Fraction of Large-Window Users

In this subsection, we study the impact of the fraction of large-window users on the performance of dynamic window resizing schemes. We simulate the random window resizing. We select the values of λ and μ so that the average resizing interval is 500 seconds, and the fraction of large-window users varies from 20% to 70% (i.e. $\lambda/(\lambda+\mu) = 20\%$ to 70%).

Figure 3.12 shows the average bandwidth consumption of each scheme. We can see that except QFS, all schemes consume more bandwidth as the fraction of large-window users increases. We can also see that the bandwidth difference between different schemes becomes smaller as the fraction of large-window users increases.

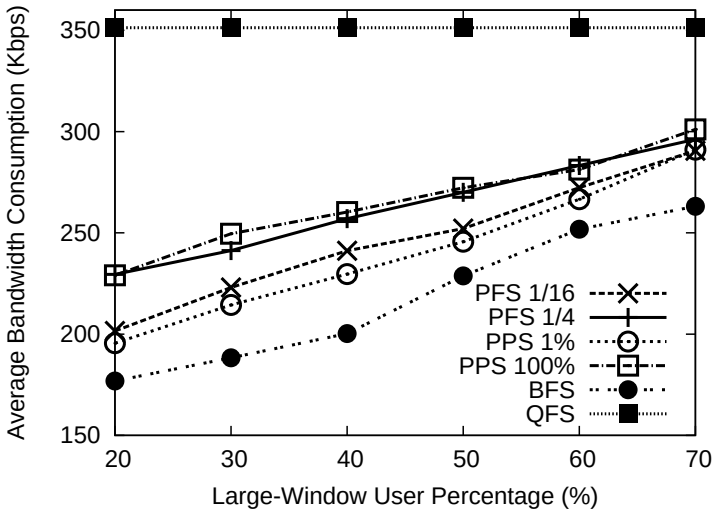

Figure 3.12: The bandwidth consumption difference between different schemes becomes smaller as the fraction of large-window users increases.

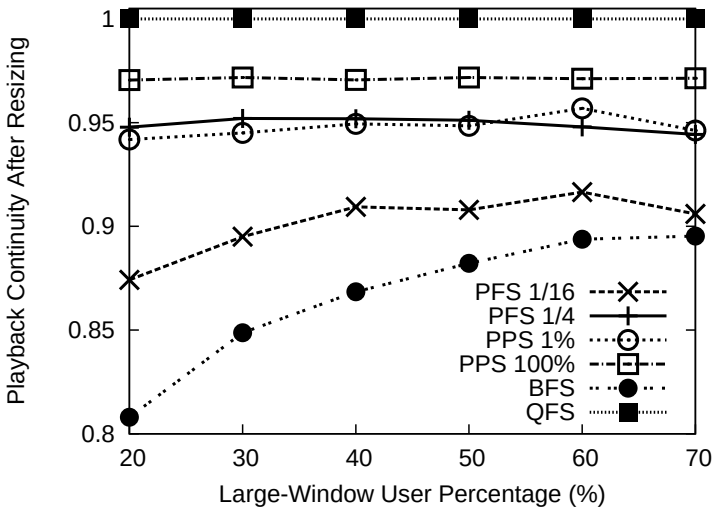

Figure 3.13: PFS with the recommended X (i.e., 4) and PPS can maintain short resizing delay independent of the large-window user fraction.

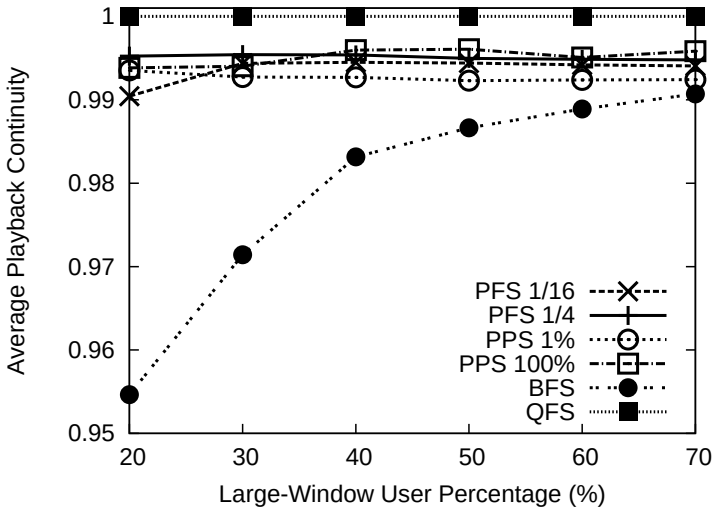

Figure 3.14: When there are a significant number of large-window users (like $\geq 50\%$), BFS can also achieve good streaming quality.

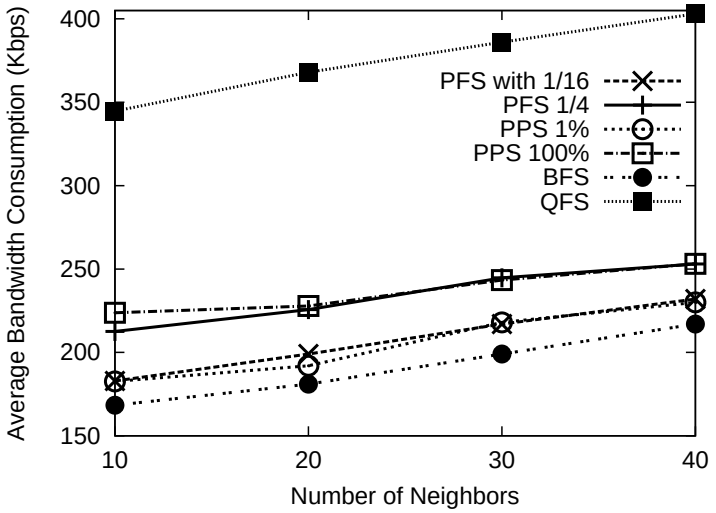

Figure 3.15: More neighbors lead to more control overhead, and thus higher bandwidth consumption.

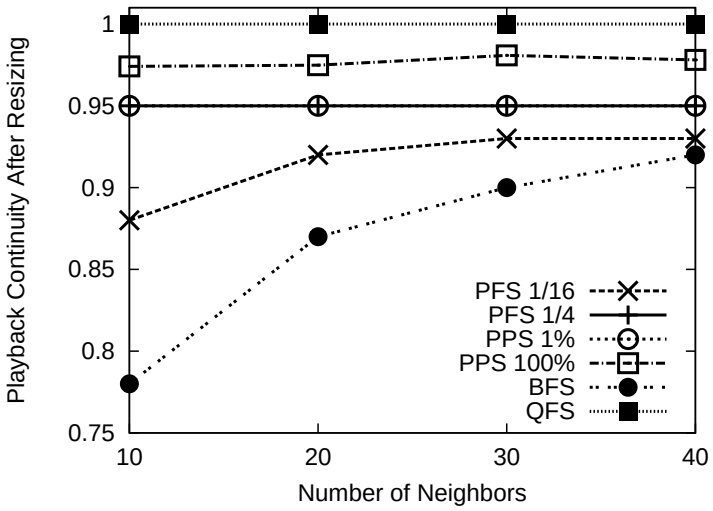

Figure 3.16: PFS with the recommended X and PPS achieve short resizing delay for any number of neighbors.

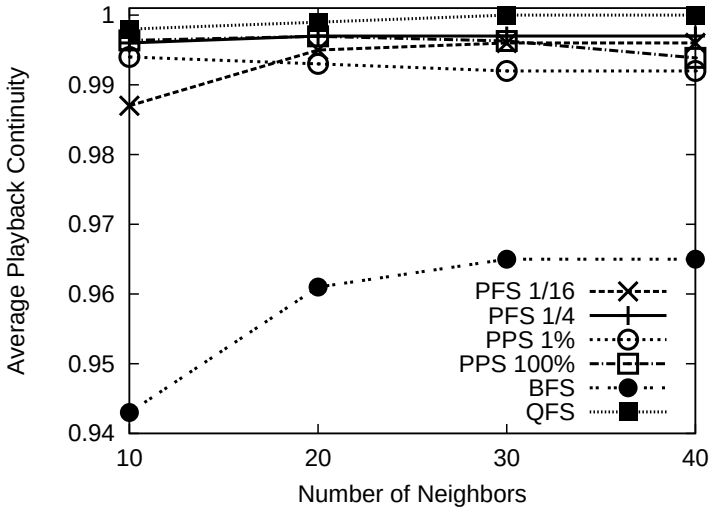

Figure 3.17: All PFS and PPS schemes achieve streaming quality very close to QFS, and much better than BFS.

This is because the bandwidth difference between different schemes depends on the number of small-window users.

Figure 3.13 shows the playback continuity between 1.5 and 2 seconds after a small-window user starts to resize to a large window. We can see that *parameter $\tilde{P}_S$ of PPS is more insensitive to the fraction of large-window users than parameter X of PFS*. Because PPS with both $\tilde{P}_S = 100\%$ and 1% can maintain short resizing delay for different fractions of large-window users. However, PFS with $X = 16$ has long resizing delay when the fraction of large-window users is small.

Figure 3.14 shows the average playback continuity of all users in the whole simulation. We can again see that all PFS and PPS schemes achieve streaming quality very close to QFS, and better than BFS. We can also see that when there are a significant number of large-window users (like more than 50%), BFS can also achieve good streaming quality.

3.5.3 Impact of the Number of Neighbors

We evaluate the impact of the number of neighbors on the performance dynamic window resizing schemes. As there are more neighbors, a small-window user resizing to a large window is more likely to find large-window neighbors and can more quickly get the enhancement layer. However, as the number of neighbors increases, the number of control packets also increases, which in turn consumes more bandwidth. We simulate random window resizing with an average resizing interval of 500 seconds and 30% of large-window users. The average number of neighbors varies from 10 to 40.

Figure 3.15 shows the average bandwidth consumption of each scheme. We can see that more neighbors lead to more control overhead, and thus higher bandwidth consumption.

Figure 3.16 shows the playback continuity between 1.5 and 2 seconds after a

small-window user starts to resize to a large window. We can see that Figure 3.16 shows a similar trend as Figure 3.13. *Parameter $\tilde{P}_S$ of PPS is more insensitive to the number of neighbors than parameter X of PFS*, and PFS with X greater than the recommended value has long resizing delay when the number of neighbors is small.

Figure 3.17 shows the average playback continuity of all users in the whole simulation. We can again see that all PFS and PPS schemes achieve streaming quality very close to QFS, and much better than BFS. We can also see that even when there are a significant number of neighbors (such as 40), the streaming quality of BFS is still much worse than that of other schemes.

Overall, we can see that increasing the number of neighbors can reduce the resizing delay and improve the streaming quality of BFS and PFS with a large X (e.g., 16), however, at a cost of higher bandwidth consumption. For PFS with the recommended X (i.e., 4) and PPS with any $\tilde{P}_S$, it is better to maintain a not very large number of neighbors. Because a very large number of neighbors increase the bandwidth consumption but do not improve the streaming quality.

3.5.4 Impact of Synchronized Window Resizing

In this subsection, we study the impact of synchronized window resizing. We simulate a P2P streaming system, which initially has 20% large-window users and 80% small-window users. No user changes its window size during the first 500 seconds. At second 500, P_S percent of small-window users simultaneously change to a large window, and all large-window users do not change their window size. We vary P_S from 20% to 100%. For example, if $P_S = 100\%$, all small-window users change to a large window at second 500.

Figure 3.18 shows the playback continuity between 1.5 and 2 seconds after the resizing event (i.e., between second 501.5 and second 502). We can see that *parameter*

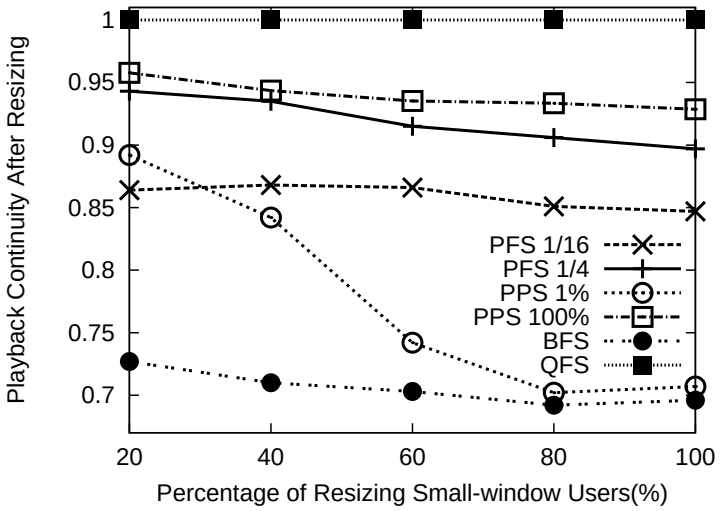

Figure 3.18: Both PPS with $\tilde{P}_S = 100\%$ and PFS with the recommended $X = 4$ achieve good playback continuity even in the extreme case with $P_S = 100\%$.

$\tilde{P}_S$ of PPS is more sensitive to P_S than parameter X of PFS. PFS schemes with both $X = 4$ and $X = 16$ can maintain similar resizing delay with different P_S's. However, the playback continuity of PPS with $\tilde{P}_S = 1\%$ drops very quickly as P_S becomes much larger than $\tilde{P}_S$. This implies that PPS with $\tilde{P}_S$ significantly lower than the actual P_S cannot achieve good streaming quality.

3.6 Conclusion and Discussion

In this chapter, we give a comprehensive study about dynamic window resizing. We formulate the problem as minimizing bandwidth consumption but still achieving short resizing delay, and maintaining good streaming quality in case of dynamic window resizing. In order to design simple and efficient P2P streaming systems for dynamic window resizing, we study two schemes which represent totally two different design philosophies - load is balanced and is distributed to all peers (PFS) and load is focused on super peers (PPS). PFS is very simple to implement and can achieve good stream-

ing quality. PPS is more complicated than PFS but it can achieve better streaming quality than PFS while consuming the same amount of bandwidth as PFS. In addition, PFS and PPS are suitable for different types of P2P streaming systems. The parameter setting of PFS is sensitive to some dynamic characteristics (e.g., dynamic resizing intervals, dynamic fractions of large-window users, and dynamic numbers of neighbors) but the dynamic fractions of simultaneous resizing users, whereas the parameter setting of PPS is insensitive to most dynamic characteristics but the dynamic fractions of simultaneous resizing users. Therefore, PFS is more suitable for P2P streaming systems with dynamic fractions of simultaneous resizing users, and PPS is more suitable for other dynamic P2P streaming systems.

Chapter 4

A General Framework for Analyzing P2P Streaming Systems with Heterogeneous User Demands on Video Quality

In this chapter, we make a further study about how to provide all users with uninterrupted video with their desired qualities in case that their demands dynamically change. The adaptive streaming rate technique is a promising method. However, in providing adaptive streaming rate services, P2P live streaming design faces the following challenges: adequate resource availability and neighborhood resource availability.

To shed more light on this problem, we propose a theoretic model to tackle the demand heterogeneity problem. With this model, we systematically study the methodology of sustaining users' heterogeneous demands with low bandwidth consumption. In this chapter, we formulate the system with a linear programming model based on

demand-supply relation considering the content and bandwidth limitations at peers. With the optimization methods, we figure out the optimal bandwidth allocation with the minimum bandwidth consumption. We investigate the system structure and propose a tiered overlay and rice-rated mechanism to quickly accommodate peers' demand changes.

4.1 Problem Formulation

Demand heterogeneity on video quality has its own characteristics which complicate distributing the uninterrupted video to users with expected qualities in the following two aspects: (1) Adequate resource availability. Demand changes of peers may break the original balance between the resource demand and the resource supply. The system should always provide enough resource supplies, otherwise it suffers bad performance. (2) Neighborhood resource availability. If peers' changing demands can be served by other peers close to them, they can see the video with their desired qualities immediately after the changes. Unfortunately, there are no studies in the literature offering a comprehensive and analytical work.

Then, what constitutes a proper solution for P2P live streaming systems in case of demand heterogeneity? We are interested in three important metrics which jointly define the performance: (1) bandwidth cost: the total amount of bandwidth consumed by the system; (2) delay: the interval between the demand changing time and the time when a peer can completely receive all required content; (3) capability: the system should support peers with heterogeneous demands on video quality as many as possible. But these three metrics usually conflict with one another, and it is challenging to solve all of them concurrently.

In most of the existing P2P streaming systems [58, 60, 70], peers all receive the

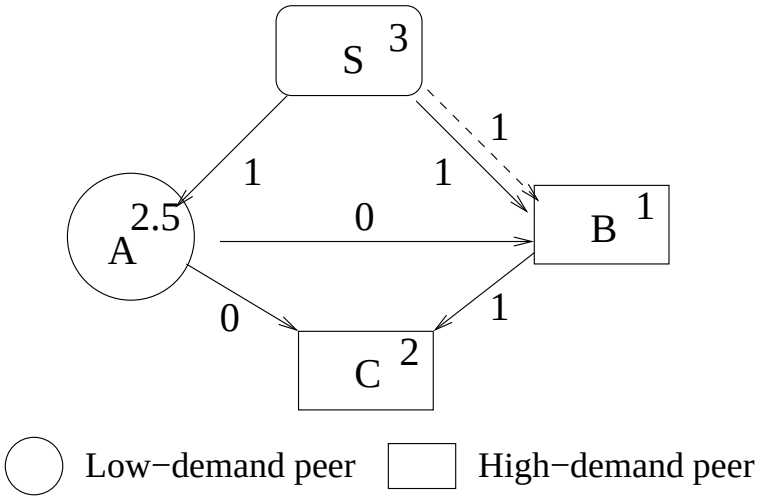

Figure 4.1: An improper upload bandwidth allocation at peers makes the system suffer heavy server load and poor performance. Superscripts of the server and peers indicate their upload bandwidth, and the streaming rate of each link is also marked next to the link.

identical content at the same streaming rate. Here, we propose adaptive streaming to solve users' demand heterogeneity on video quality. There are two common coding methods to provide adaptive streaming, namely Multiple Description Coding (MDC) [17] and Layered Coding [16]. We use layered coding since it has noticeably higher coding efficiency than MDC, but our work can easily be extended to MDC. With layered coding, peers may receive different layers corresponding to their different demands. The peers with low demands on video qualities only need to receive a subset of the layers, so the bandwidth consumption is reduced.

As peers have different layers according to their demands with adaptive streaming rates, their upload bandwidth allocation among these layers should be well planned. Consider the simple scenario in Figure 4.1, where the video is encoded into one base layer at a rate of 1 segment/s and one enhancement layer also at a rate of 1 segment/s. The server utilizes all its bandwidth to steam the video to peers A and B, and the connection from A to B becomes useless as B already has both layers. Peer C is

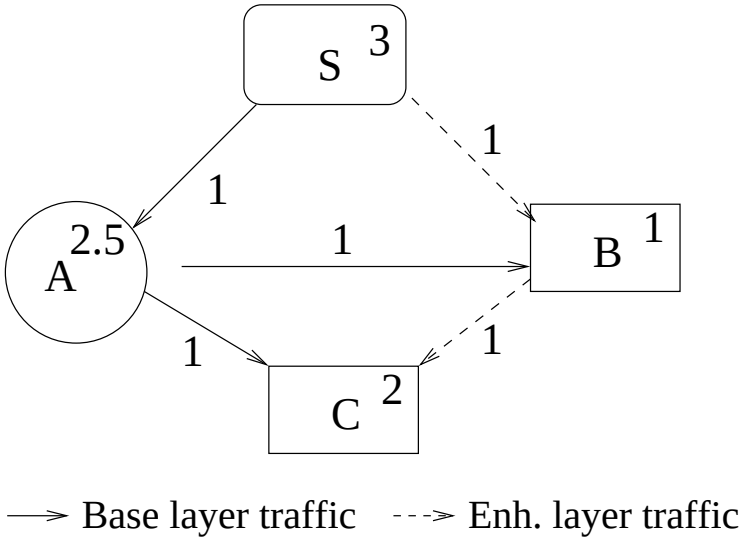

Figure 4.2: A proper upload bandwidth allocation reduces servers load and improves systems satisfaction under the same settings. Peer A streams the received base layer to peers B and C, so the bandwidth of server S and peer B can be dedicated to the enhancement layer.

unsatisfied as none of its neighbors forwards the enhancement layer to it - peer B's upload bandwidth is used to stream the base layer and peer A does not have the enhancement layer.

The above problem can be solved by reallocating the participating peers' upload bandwidth. As shown in Figure 4.2, peer A streams its received base layer to peers B and C, so that 33% of the server S's bandwidth is saved, and peer C's demand on high-quality video is also satisfied. For this problem, the main challenge lies in determining how to optimally allocate the upload bandwidth to layers at peers in order to minimize the server load as well as to improve the system satisfaction.

The challenge is not easy to overcome in practice. First, sometimes bandwidth allocation can achieve the optimal bandwidth efficiency, but it cannot solve the whole problem. In Figure 4.3, peer B's upload bandwidth decreases to 0.5, for example, if other applications also run at peer B and consume part of its bandwidth. No

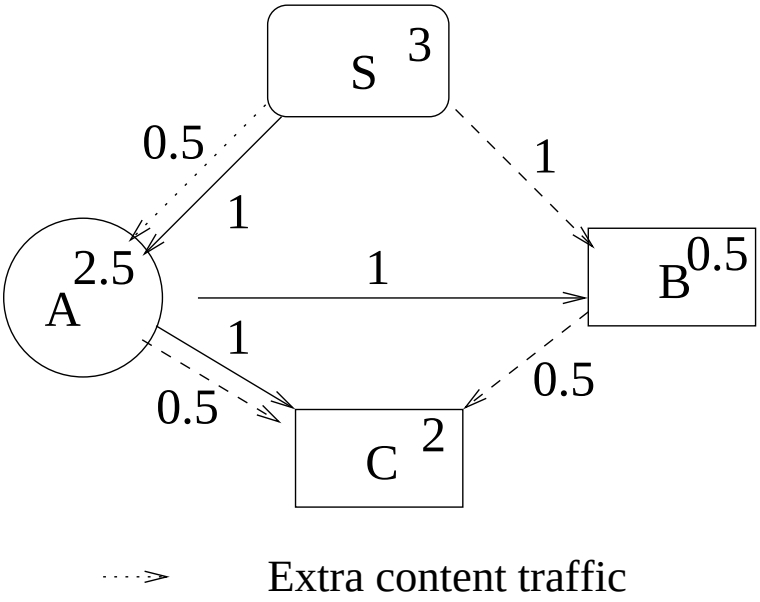

Figure 4.3: Bandwidth allocation cannot always solve the problem, and extra forwarding is required. Peer A downloads the enhancement layer not required by itself, then streams it to peer C, as C cannot get the enhancement layer from B at a sufficient streaming rate.

matter how complicated a bandwidth allocation is designed, it is impossible for peer C to get both the base layer and the enhancement layer. As shown in Figure 4.3, peer A additionally downloads the enhancement layer and forwards it to peers B and C. Peer A serves as a relay to stream the enhancement layer to peer C, so all peers in the system can receive the video with their expected qualities. Second, users dynamically change their demands which subsequently change the layers owned by peers. A bandwidth allocation which works well now may not work in the future.

Usually, the more choices on video qualities that the system provides, the more bandwidth can be saved. This is because the demands can be differentiated more precisely. But in practical implementation, the complexity also increases, and coding efficiency decreases.

4.1.1 Model

To study this problem more precisely, we define the following symbols and definitions.

- Let N be the set of all peers, and n be the number of these peers.
- As there exist diverse demands on video quality, we set up thresholds to differentiate them into corresponding viewing groups. Each type of video quality demand has its corresponding view group. This may lose some precision on demand representation, but is still capable to grab the essence of demand heterogeneity. Let J be the number of groups in the system, $M_j(t)$ represents the set of peers of view group j, and $m_j(t)$ represents the number of peers in view group j. There are total n peers. Then the state of the system at any given time can be defined by a J-dimensional vector $M(t) = (M_1(t),\ \ M_2(t),\ \ ...,\ \ M_J(t))$. Since the whole system is dynamic and we focus on the balance between the resource demand and the supply, we remove t from the notation to simplify the description. Please note that the vector changes over time. Without loss of the generality, we consider that M_1 represents the view group with the lowest demands defined in the system, while M_J represents the view group with the highest demands. Obviously, the groups are dynamic as peers can change their demands based on their own preferences.
- Let R be the full streaming rate of the channel watched by these n peers. With layered coding, the stream is divided into J layers. The number of layers does not necessarily equal to the number of demands provided by the system, but here we make them be equal to simplify our description. Peers in view group j request layers from layer 1 to layer j. r_j represents the streaming rate of layer j, and $R = \sum_{j=1}^{J} r_j$, so the required streaming rate of view group j is $R_j = \sum_{i=1}^{j} r_i$. For example, in 2-layer layered coding with streaming rates of

110 kbps and 220 kbps respectively, peers in view group 1 receive the base layer of the video at 110 kbps, and peers in view group 2 are watching the same video with high definition at rate 330 kbps.

- Let V be the server's upload bandwidth. Let V_j denote the upload bandwidth of the server allocated for layer j. Note that, V_j is a variable with $r_j \leq V_j$ and $\sum_{j=1}^{J} V_j = V$.

- Let u_i be the upload bandwidth of peer i, while u represents peers' average bandwidth. To simplify the description, we assume that there are two types of bandwidth: u_h and u_l, with $u_h > R > u_l$. The percentage of high bandwidth peers is α.

Below we provide some definitions to more precisely measure system-wide satisfaction.

Definition 1 A peer is satisfied if it receives all layers which it requires and the received rates of these layers are equal to their streaming rates.

This definition is straightforward, but we want to emphasize that receiving any further layers after gathering all required layers, does not improve user satisfaction as they already can view the video with their desired qualities.

Definition 2 Layer J is fully served by the system if: $D_i \geq B_j - O_j$, where B_j, D_j are the total amount of the supply and the demand for layer j, respectively. O_j is the control overhead for layer j.

When the system can provide the supply of layer j more than the demand, we consider layer j as fully served. Thus, the probability of layer j's satisfaction is defined as:

$$P_j = P(D_j \geq B_j - O_j)$$

Further, if layers from 1 to J are all fully supported, we say that the system sustains all peers with their heterogeneous demands. The system-wide satisfaction probability across all layers is defined as:

$$PS = \prod_{j=1}^{J} P(P_j|P_{j-1}),\ P_0 = 1 \tag{4.1}$$

Please note that the system is satisfied only if all the J layers have enough supplies compared to their demands. This formula is used to measure the impact of bandwidth and demand distribution on system satisfaction.

Definition 3 If a peer can get all required content in a short bounded period T after it changes its demand, we say that short delay is achieved.

Usually, a peer degrades its demand by stopping requests for some layers, while it upgrades the demand by starting requests for more layers. In the first case, its changed demand can be immediately satisfied, while in the second case, some buffering time may be required before playing the stream of the upgraded demand.

4.2 Resource supply and demand

Based on predefined demand differentiating methods, peers in the system are divided into J groups: from the peers in M_1 with the lowest stream demand to high-definition peers in M_J with $\sum_{j=1}^{J} m_j = n$. Peers in M_1 receive the first layer-the base layer, while peers in M_2 receive 2 layers-the base layer and the first enhancement layer, ...,

peers in M_J receive the base layer and $J-1$ enhancement layers. Peer i in view group j allocates u_i to layers from 1 to j, and its contribution to layers higher than j is 0 because it does not have these layers.

The basic requirement of the system is to ensure that for each layer, the bandwidth supply exceeds or equals the bandwidth demand. To simplify the description, if the supply of layer $j > 1$ cannot support its demand, we call it a lean layer. If there is no lean layer in the system, the whole system is satisfied as it has enough supplies for all layers. The population of peers is distributed as $m_1, m_2, ..., m_J$. So we can easily get the total bandwidth demand for layer j, $j \in [1, J]$ as:

$$D_j = \sum_{i=j}^{J} m_i * r_j \tag{4.2}$$

In the following subsection, we investigate what an optimal bandwidth allocation is.

4.2.1 Optimal bandwidth allocation

Let U_j represent the total amount of bandwidth of peers in view group j, $U_j = \sum_{i \in M_j} u_i$. Note that U_j is shared by layers from 1 to j. All peers in view group $i \geq j$ have layer j, so the layer j's bandwidth is in the range of $B_j \in (0, \sum_{i=j}^{J} U_i + Vj]$. As the total amount of bandwidth for these layers cannot exceed the total upload capacity, we have $\sum_{i=j}^{J} B_i \leq \sum_{i=j}^{J} (U_i + V_i)$ for $j \in [1, J]$.

As shown in the aforementioned example, there may exist $D_j > B_j$ for some $j \in [1, J]$. We use ΔB_j to denote the bandwidth shortage for layer j. Namely, $\Delta B_j = max\{0, D_j - B_j\}$. To simplify the description, we ignore the control overhead O_j as it is relatively small and usually is a fixed fraction of D_j. We have $B_j + \Delta B_j = D_j$

for $j \in [1, J]$.

We want to minimize the amount of these ΔB_js by efficiently utilizing existing resources, so that the gap between the demand and the supply is small. We formulate this allocation problem as the following optimization problem.

$$
\begin{aligned}
&minimize \sum_{j=1}^{J} \Delta B_j, \ s.t. \\
&B_j + \Delta B_j \geq D_j \\
&and \\
&\sum_{i=j}^{J} B_i \leq \sum_{i=j}^{J} (U_i + V_i), \ for \ j \in [1, J] \\
&\Delta B_1 \geq 0, \Delta B_2 \geq 0, \ ..., \ \Delta B_J \geq 0 \\
&B_1 \geq 0, B_2 \geq 0, \ ..., \ B_J \geq 0
\end{aligned} \tag{4.3}
$$

The first constraint in Eq. (4.3) shows that the system needs to make the supply of layers from 1 to J greater than or equal to the demand. The second constraint defines the scopes of variables B_js.

This is a linear program as well as a convex program. The optimal ΔB_js can be solved using Karush-Kuhn-Tucker Theorem [56]:

$$
\begin{aligned}
\Delta B_j^* = max \left\{ \sum_{i=j}^{J} (m_i^* \sum_{k=j}^{i} r_k - U_i - \Delta B_{i+1}^*) - V_j, 0 \right\}, \\
for \ \Delta B_{J+1}^* = 0
\end{aligned} \tag{4.4}
$$

The expression (4.4) of ΔB_j^* shows that peers should allocate bandwidth to their high layers with high priorities. In layered coding, peers which have layer j also have layers lower than j. Lower layers are more important than higher layers in layered

coding. When there exists a lean layer $i < j$ with $\Delta B_i > 0$, we convert layer i to a non-lean layer through making peers release ΔB_i which is allocated to layers higher than i and assigning the released bandwidth to layer i. Bandwidth for layer j decreases by ΔB_i, but the total amount of $\sum_{i=1}^{J} \Delta B_i$ is still the same. So in practice, *we make peers serve low layers first, but always try to allocate more bandwidth to high layers.*

4.2.2 Is bandwidth allocation alon enough?

However, the optimal bandwidth allocation may not always be able to sustain all peers with their desired demands as shown in Figure 4.3. In this subsection, we explore when and why the optimal bandwidth allocation fails to sustain all peers with their demands.

The essence of bandwidth deficiency of layer j is the unbalanced relation between the demand of layer j and its supply. Assume that α percent of peers have high bandwidth u_h and the remaining peers have low bandwidth u_l . Layer j becomes a lean layer if there are not enough high bandwidth peers contributing their bandwidth to layer j. Let x_j denote the number of peers with u_h in view group j, thus the potential available bandwidth of view group j can be expressed as $U_j = x_j * u_h + (m_j - x_j) * u_l$. Please note that α only determines the total number of high bandwidth peers in the whole system, but the number of high bandwidth peers in each view group is random. Now we consider under what conditions layer j can be supported with the optimal bandwidth allocation, namely $\Delta B_j = 0$. Following Formula (4.4), we have

$$\sum_{i=j}^{J} U_i \geq \sum_{i=j}^{J} (m_i * \sum_{k=j}^{i} r_k) - V_j$$

Replace U_i with $U_j = x_j * u_h + (m_j - x_j) * u_l$, and we have

$$\sum_{i=j}^{J} x_i \leq \frac{\sum_{i=j}^{J}(m_i * \sum_{k=j}^{i} r_k - m_i * u_l) - V_j}{u_h - u_l}$$

Let $\beta_j = \frac{\sum_{i=j}^{J}(m_i * \sum_{k=j}^{i} r_k - m_i * u_l) - V_j}{u_h - u_l}$. Namely, only if there are at least β_j can layer j have the enough supply. Thus, the probability of layer j's satisfaction is:

$$P_j = P(\sum_{i=j}^{J} x_i \geq \beta_j) \tag{4.5}$$

Using Eq. (4.1), we can obtain the probability of system satisfaction.

Whether the system is able to support all peers' demands is also affected by the demand distribution. For a given system with a fixed demand distribution, the value of P_j is affected by the percentage of high-bandwidth peers in the system. On the other hand, when the percentage of high-bandwidth peers in the system is stable, the demand distribution determines the value of P_j.

Let us consider a system which consists of 1,000 peers with two types of bandwidth, and there are high, medium and low demands. The video is encoded into three layers with identical streaming rate at $\frac{R}{3}$, so the peers with the three types of demands need to receive 1, 2 and 3 layers, respectively. We set $u_h = 2R$ and $u_l = 0.2R$ which are reasonable according to practical settings. For example, typically the streaming rate of a video provided by existing P2P streaming systems is about 300-500 kbps, and a residential user may only have 100 kbps upload bandwidth while a campus user may have high upload bandwidth like 1 Mbps. α is used to control the percent of high-bandwidth peers. The number of peers' neighbors is 30-50, as typically used in

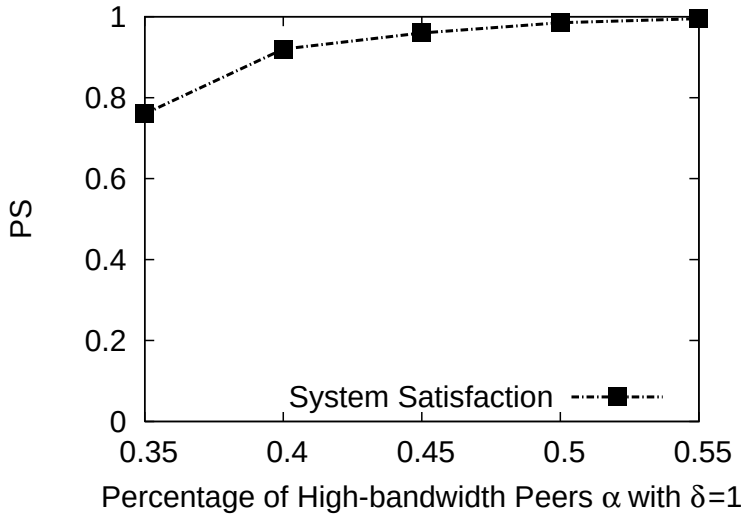

Figure 4.4: The higher percentage of high-bandwidth peers is, the easier for the system gets satisfied.

prevailing P2P streaming systems [70, 58]. The peers' demand distribution follows the Zipf law [1]. Namely, $\frac{m_j}{m_{j+1}} = \delta$, where δ is the Zipf parameter. We plot the results of satisfaction probabilities of the system under different bandwidth settings and demand distributions.

Firstly, we check the impact of bandwidth distribution in Figure 4.4. The demand distribution follows Zipf law with $\delta = 1$. The results show that the more high bandwidth peers are in the system, the higher probability of system satisfaction is. When the system has many high-bandwidth peers (like 50%), it can always get satisfied. But there are not so many high-bandwidth peers in real world to make all layers always have enough supplies.

Next, we check the impact of demand distribution on the probability of system satisfaction. We choose a fixed $\alpha = 0.45$ to ensure that the system has sufficient bandwidth even if all peers have high video quality demands. Figure 4.5 shows that a system with heterogeneous demands gets a high probability of satisfaction only for $\delta \leq 1$. When δ becomes larger, the demand is more unevenly distributed. For

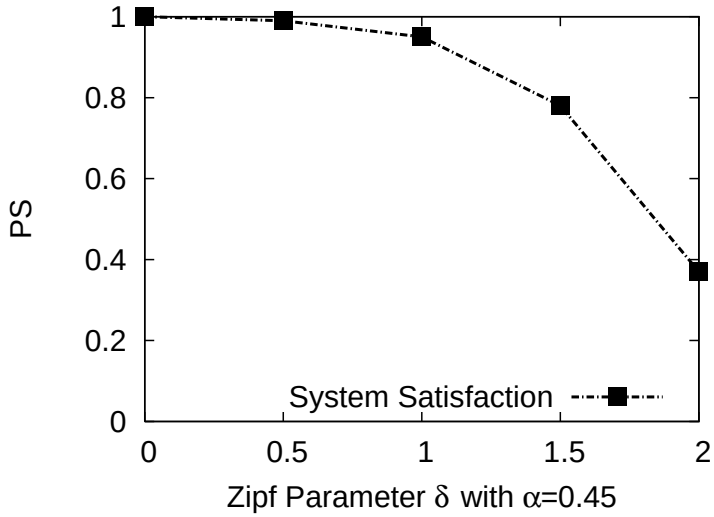

Figure 4.5: Uneven demand distribution decreases the probability of system satisfaction, even with sufficient bandwidth.

example, high demand peers become rare when $\delta \geq 1$, and the small population of peers owning layers 2 and 3 makes the system hard to find sufficient bandwidth for them. While $\delta < 1$, there are many peers having these layers, so the supplies of them are no longer a problem. The results show that the system needs to maintain a considerable number of peers having high layers.

Sometimes extra efforts are required. Without increasing server load, peers help one another by forwarding extra data not required by themselves. The number of these peers should be as few as possible to limit the bandwidth consumption, and we discuss how to select them in the following subsection.

4.2.3 Peer selection policy

As each selected peer also downloads a copy of a lean layer, we need to select peers as few as possible to minimize the bandwidth consumption. Given a candidate peer i for a lean layer j in view group $k(k < j)$, bandwidth allocated for layer j is limited by

its spare bandwidth u'_i. Only peers with $u'_i > r_j$ will be chosen to forward the extra layer j, and amplify bandwidth for layer j by $u'_i - r_j$ [80]. Otherwise, the gap between the demand and the supply of layer j increases rather than decreases as these peers consume more bandwidth than their contribution. Let H_j represent the set of chosen peers for layer j. Our goal is to:

$$\begin{aligned} &min \sum_{j=1}^{J} \sum_{i \in H_j} r_j, \\ &subject\ to \\ &\sum_{i \in H_j} u'_i \geq \Delta B_j,\ for\ j \in [1, J] \end{aligned} \tag{4.6}$$

Formula (4.6) shows that the optimal method is to select peers with high spare bandwidth. However, this works only if peers do not reallocate their bandwidth. As peers follow the optimal method to adjust their bandwidth allocation, a peer with the maximum spare bandwidth now may not have the maximum bandwidth contribution in the future.

Theorem 1 *From peers in view group$k$($k < j$), selecting those with the highest bandwidth in group k can minimize the extra bandwidth consumption.*

Proof We assume that there exist two candidates peers a and b in view group k with $u_a < u_b$. We denote the total spare bandwidth of peers in group k with U'_k . Under optimal bandwidth allocation, peers allocate their bandwidth to higher layers as much as possible after serving the lower layers. Their bandwidth contribution is constrained by: the selected peer's bandwidth and the total spare bandwidth of group k. The first constraint is easy to understand. The second constraint is because peers need to firstly serve layers lower than j in the optimal bandwidth allocation. Thus, for the two candidates, the potential increased bandwidth for layer j is $\min(u_a, U'_k)$ and

$\min(u_b, U'_k)$, respectively, and $\min(u_a, U'_k) \leq min(u_b, U'_k)$. So to get the maximized bandwidth contribution, the highest bandwidth peers should be selected first.

4.3 Framework for accommodating video quality demand heterogeneity

In this section, we propose a framework to implement the ideas derived in Section 4.2 and to achieve short delay when peers change their demands. We present the tiered structure to organize peers into cooperative groups. The distributed algorithms to implement the efficient bandwidth allocation and peer selection within a cooperative group are also discussed.

4.3.1 Tiered P2P streaming

It is hard to find an efficient solution to stream the video to peers with their desired demands. Consider the case in Figure 4.6: the video is encoded into two layers, and peers have different numbers of layers based on their own interest. As only a subset of all n peers needs layer 2, streaming the content of layer 2 to these peers in an efficient way requires finding a Steiner tree in a weighted graph, which is NP-Complete [27].

However, we can organize peers into several cooperative groups, as shown in Figure 4.6. The server maintains the bandwidth information of these groups. When a new peer joins, the server determines which cooperative group the peer is assigned to based on its bandwidth. Thus, all cooperative groups have similar bandwidth capacities. After reorganizing them into small groups, each cooperative group selects one peer to join the higher tier. In Figure 4.6, peers are organized into two tiers: tier-1 and tier-2, and one peer in each cooperative group is selected to join the higher tier - tier 1. In a cooperative group, the tier-1 peer requests all layers of the video, so it is possible

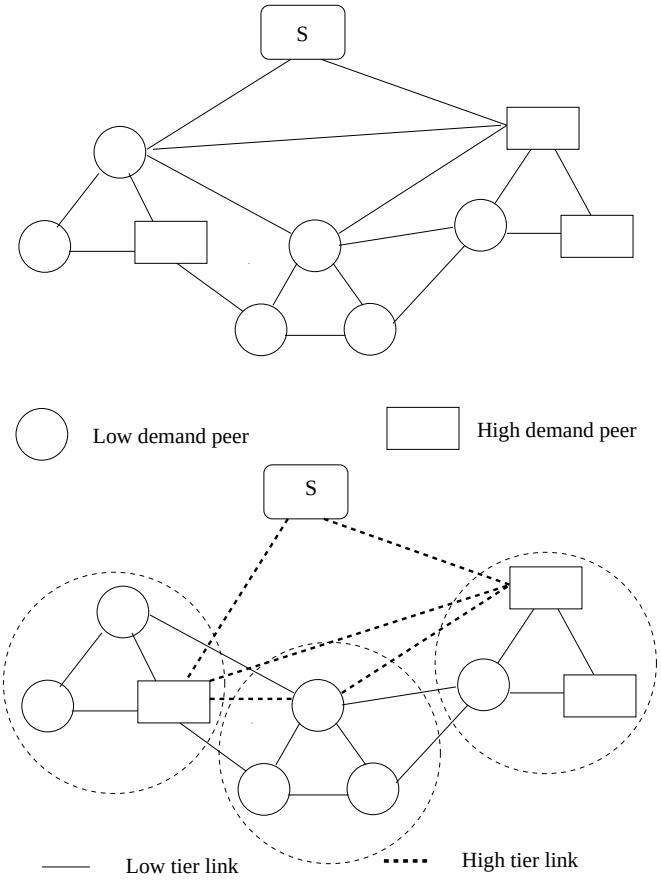

Figure 4.6: Tiered overlay of a P2P streaming system

for the other peers to obtain all layers in the local scope.

All the tier-1 peers as well as the server constitute a mesh, which is used in most of commercial P2P live streaming systems. This mesh distributes all layers of the video. Even if sometime these tier-1 peers do not have high video quality demands, they still get rewards as tier-1 peers. Compared to other peers, they are closer to the server, and have shorter playback delay. Please note that the connections between peers from different cooperative groups also exist, but the peers always serve the neighbors in the same cooperative group with higher priorities. Usually only the peers with high bandwidth and high demands are more likely chosen to be tier-1 peers.

Cooperative groups are orthogonal to view groups. Peers are organized into cooperative groups based on their bandwidth, while they are divided into different view groups according to their demands on video qualities. As peer demands are orthogonal to their bandwidth, these two groups are orthogonal.

Let τ denote the size of a cooperative group, we set up two parameters: τ_{min} and τ_{max} representing the minimum and maximum size of a cooperative group, respectively. Usually peers maintain dozens of connections in P2P live streaming systems [94]. For example, the default number of connections is 18 in UUSee [70]. Accordingly, we set the cooperative group size to be between 30-50, and peers inside a cooperative group maintain a neighbor list of size 15. Under these settings, peers are directly connected with 30-50% peers in the cooperative group, so they can easily find layers that they need within their own groups. Even in the worst case that peers need to find new neighbors after changing their demands, they can quickly complete searching all group members. For a tier-1 peer, it also has connections with other tier-1 peers, so its neighbor list is about twice that of other peers' in the cooperative group.

We claim that there exist enough stable peers which can be chosen as tier-1 peers. The cooperative group size is between 30-50 and there is one tier-1 peer in each

cooperative group. Namely, there is one tier-1 peer in every 30-50 peers, and thus the percentage of tier-1 peers roughly is 2-3%. A recent systematic study of a real large-scale system shows that the ratio of stable peers is 73.8-85.0% per snapshot or 5.5-15.4% per trace [74], which means that there are sufficient stable candidates as tier-1 peers. Further, we can assign backup peers for these tier-1 peers in the same cooperative groups, so they can quickly take the role of the tier-1 peers which are leaving.

There are several advantages of the tiered mesh structure: (1) It is easy to implement. The existing sophisticated P2P technologies can be directly used to distribute video to tier-1 peers, and the derived principles are easy to be implemented in a cooperative group, due to its relatively small scale and highly-connected feature. (2) Peers have accesses to all layers in a cooperative group, which makes peers be able to immediately receive all required content after the demand changes. (3) Impact of churn, including both peer churn and demand churn, is restricted in a small scope.

Let us consider delays after the demand changes. As discussed before, only peers which change their demands from low to high suffer delays until they get the newly required layers. Most of the time, the tiered structure makes the peers easily find their required layers, because the cooperative group maintains enough bandwidth supplies compared to the demands for all layers.

4.3.2 Distributed algorithms

In this subsection, we introduce the price-rated mechanism to implement the following principles derived from Section 4.2:

- A peer allocates its bandwidth to a high layer as much as possible after fully serving all layers lower than that layer.

- Peers which can allocate more bandwidth to lean layers are chosen to forward extra lean layers.

Layers in a cooperative group have different prices reflecting whether the supplies are sufficient compared to the demands. For a single layer, the unit price is defined by:

$$p_j = \begin{cases} \alpha_j, \ if \ \frac{d_j}{b_j} > 1 \\ 1, \ if \ \frac{d_j}{b_j} = 1 \\ 0, \ if \ \frac{d_j}{b_j} < 1 \end{cases} \tag{4.7}$$

where d_j and b_j represent the demand and supply of layer j in a cooperative group, respectively with $\alpha_1 > \alpha_2 > ... > \alpha_J > 1$. d_j is determined by current peers' demands, while b_j is determined by the bandwidth allocation at these peers.

According to the price information of all layers, peers are able to determine when and which *lean layers* they should contribute their bandwidth to. Peers pay for downloading layers from their upstream peers, and charge their downstream peers for forwarding the layers to them. For peer i in view group j (if it forwards an extra layer $j+1$, it will be considered as in view group $j+1$) , it allocates $a_{i,k}$ bandwidth of its total u_i to layer k, and it aims to get the maximized payment.

$$\begin{aligned} &maximize \ \Phi(i) = \sum_{k=1}^{j} a_{i,k} p_k \\ &subject \ to \\ &\sum_{k=1}^{j} a_{i,k} \leq u_i \\ &a_{i,k} = 0, \ for \ k > j \end{aligned} \tag{4.8}$$

The first constraint in Formula (4.8) means peer i's contribution cannot exceed its capacity, and the second constraint means that it can only forward the layers that it has.

For all peers in a cooperative group, we have

$$\sum_{i=1}^{\tau} a_{i,j} \leq d_j, \ for\ j \in [1, J] \tag{4.9}$$

Theorem 2 *With the distributed algorithms, peers allocate their bandwidth to the high layers as much as possible.*

Proof We prove this by contradiction. If there exists a low layer j with $b_j < d_j$, while layer $k > j$ has $b_k > 0$. For a peer i which allocates a_k bandwidth to layer k, it gets profit $\Phi(i)$. If it decreases a_k by Δa_k where $\Delta a_k < b_j < d_j$ and reallocates this Δa_k to layer j, the new profit $\Phi(i)' = \Phi(i) + \Delta a_k * (p_j - p_k) > \Phi(i)$ as $p_j = \alpha_j > \alpha_k \geq p_k$. $\Phi(i)$ is not maximized.

If the high layer j is allocated $b_j \geq 0$ bandwidth which is less than the possible maximum value, there exists another bandwidth allocation with which layer j receives $b'_j > b_j$. There are two possibilities for peer i to provide the Δb_j: allocate from its spare bandwidth or decrease its bandwidth allocated for layers lower than j and reallocate that to layer j. For both of the two cases, we have $\Phi(i)' > \Phi(i)$.

4.4 Performance evaluation

In this section, we evaluate the effectiveness of the proposed framework in P2P live streaming systems in case of heterogenous video quality demands. We choose PPS

Peers' distribution #	1 Mbps(%)	384Kbps (%)	128Kbps (%)
1	10	35	55
2	10	45	45
3	15	39	46
4	18	40	42
5	20	45	35
6	23	46	31

Table 4.1: Table of bandwidth distributions

[89] as a reference and implement PPS integrating our framework called tiered PPS. In PPS peers request different number of layers according to their demands on video qualities, and some of them work as helpers to forward layers which are not required by themselves. Peers periodically collect the information about the supplies of layers in their neighborhood, and then determine to work as helpers or not. In PPS, its overlay structure and peer cooperation are not well-designed, and there is no optimization on bandwidth allocation of peers. The details of PPS can be seen in [89].

In the default settings, there are 1,000 peers in the system with an either high or low demand on video qualities. The demand distribution from low to high follows Zipf law with default Zipf parameter $\delta = 1$. The video is encoded into J layers, with the default value of two corresponding to the two types of demands. The two layers have streaming rates of $r_1 = 200$ Kbps and $r_2 = 100$ Kbps and the total rate of the video is $R = 300$ Kbps. There are three types of peers as shown in Table 4.1, and we choose no. 6 as the default setting.

We check if current demands can be sustained by calculating the playback continuity which is the ratio of packets arriving before their playback deadline. Figure 4.7 shows that the tiered PPS provides high and stable stream quality. The framework makes the system relatively stable and peers inside a group can cooperate much more

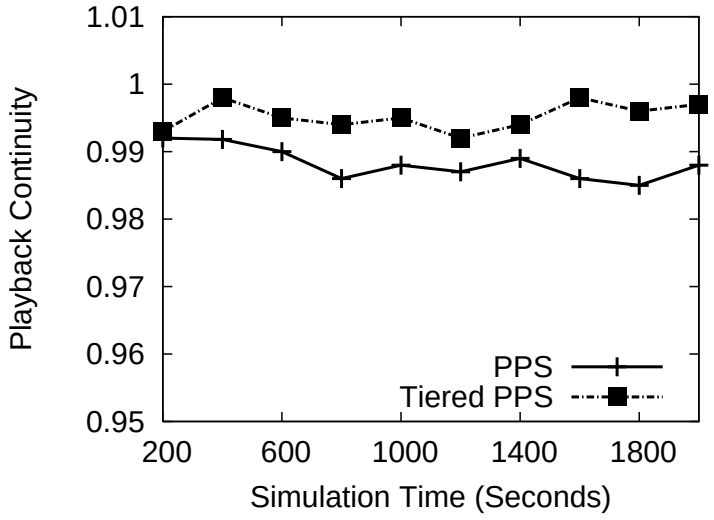

Figure 4.7: The framework improves the overall performance via stable structure and cooperation.

easily. Further, the optimal bandwidth allocation increases the bandwidth efficiency, so the peer demand is supported with higher probabilities. Then, we shift our interest to the bounded delays.

Figure 4.8 shows that shorter delays are achieved with the framework. Because the framework ensures that peers can locally find the supplies of layers that they need. Thus the changes can be immediately accommodated.

Figure 4.9 shows the impact of the framework on bandwidth efficiency. With the framework, bandwidth consumption considerably decreases, because the optimal bandwidth allocation and peer selection minimize the total amount of bandwidth to support peers' heterogeneous demands. Compared to the values before integrating the framework, the bandwidth consumption fluctuates greatly. This is because peers cooperate in a cooperative group much smaller than the whole system, so deficiency and excess of bandwidth of layers can be quickly detected, and the system can be efficiently tuned up.

Further, the tiered structure also improves the playback delay. As peers are or-

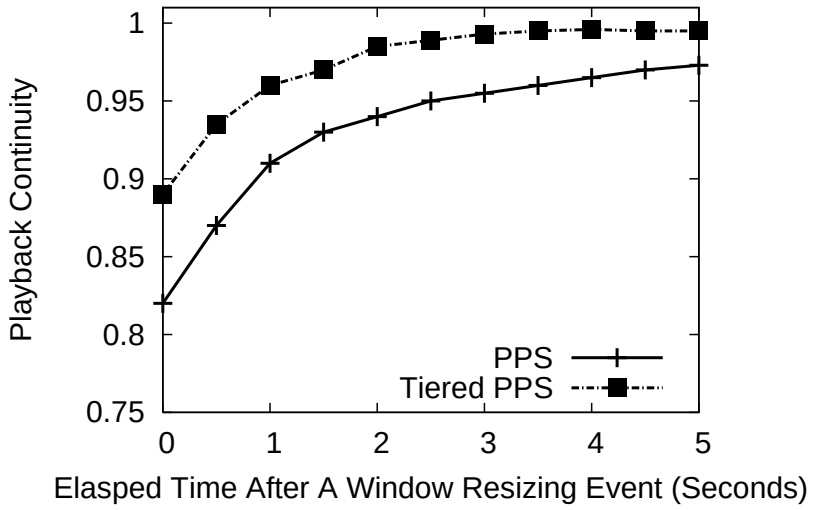

Figure 4.8: The framework considerably improves the local resource availability.

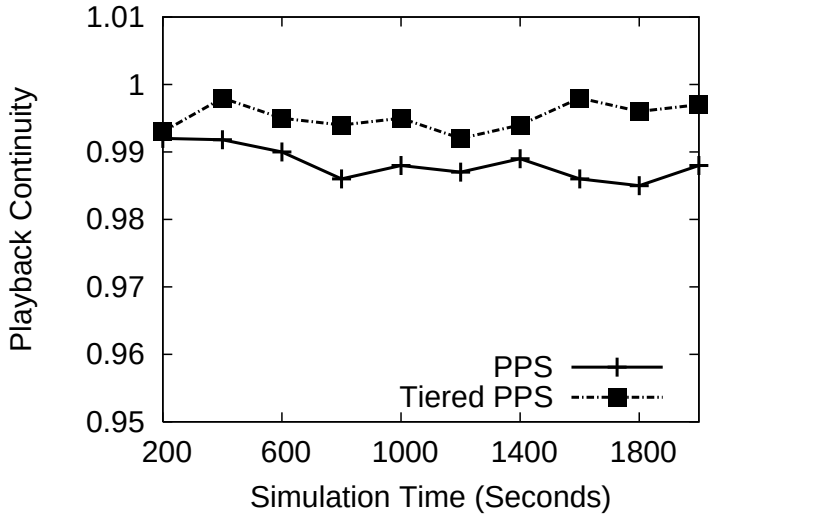

Figure 4.9: The framework enables the system achieve higher bandwidth efficiency.

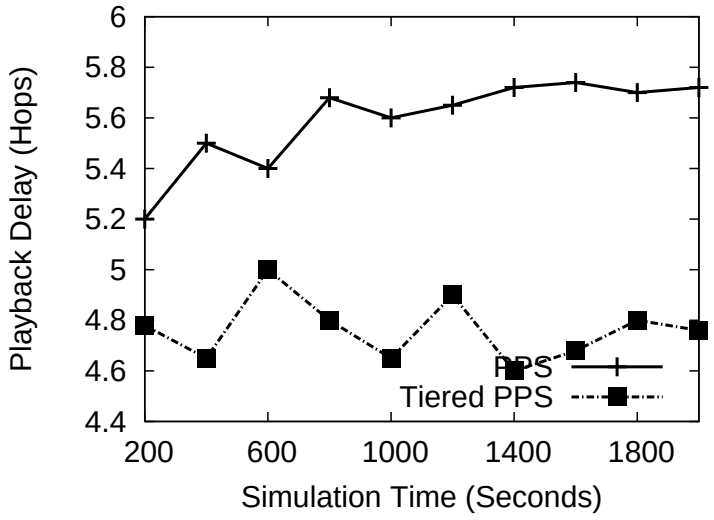

Figure 4.10: Tiered PPS has shorter playback delays.

ganized with tiered structure, the content distribution is more efficient. Figure 4.10 shows a dramatic improvement on average playback delay after using the framework.

We next evaluate the framework under various circumstances. First, we set up several close to real life configurations of peers' bandwidth distribution as shown in Table 4.1, while we maintain the demand distribution with Zipf parameter $\delta = 1$. Figure 4.11 shows that our framework always beats PPS as it improves the bandwidth efficiency as well as effectiveness of cooperation among peers. Figure 4.12 shows packet delivery ratio 2 seconds after peers increase their demands. Our framework provides peers with short delay in all the settings. Figure 4.13 depicts that our framework helps PPS reduce bandwidth consumption via improving bandwidth efficiency. All these improvements are more evident, especially when the system has a low percentage of high upload bandwidth peers.

Next we shift our interest to testing if our framework is robust under different demand distributions. We change the demand distribution with Zipf parameter δ from 0 to 2, while we maintain the 6th bandwidth distribution shown in Table 4.1.

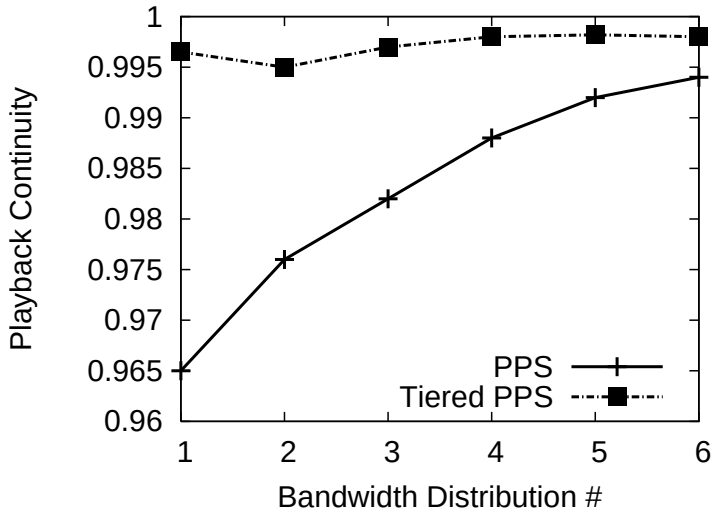

Figure 4.11: The overall performance is always improved by the framework.

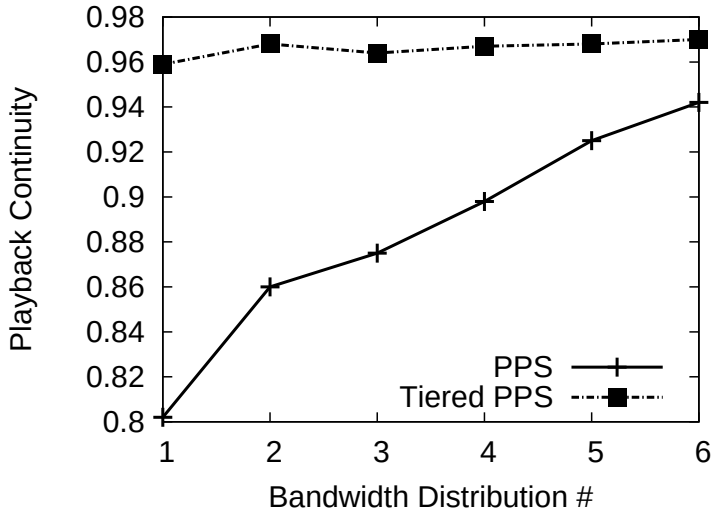

Figure 4.12: The framework enables peers get video more quickly.

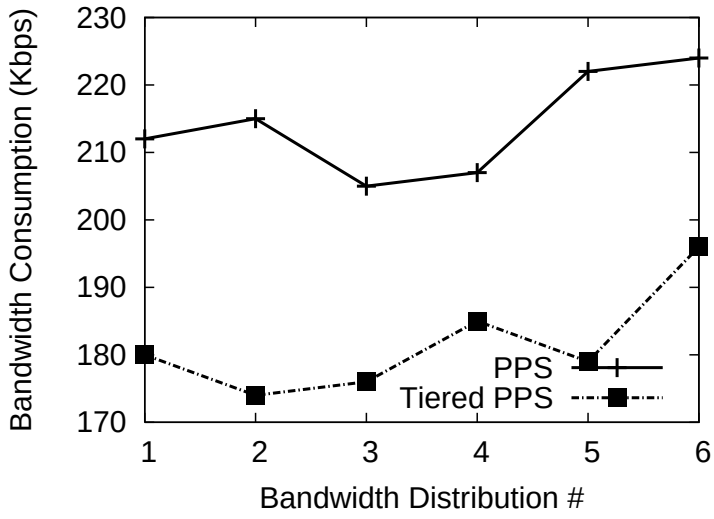

Figure 4.13: The framework helps PPS reduce bandwidth consumption.

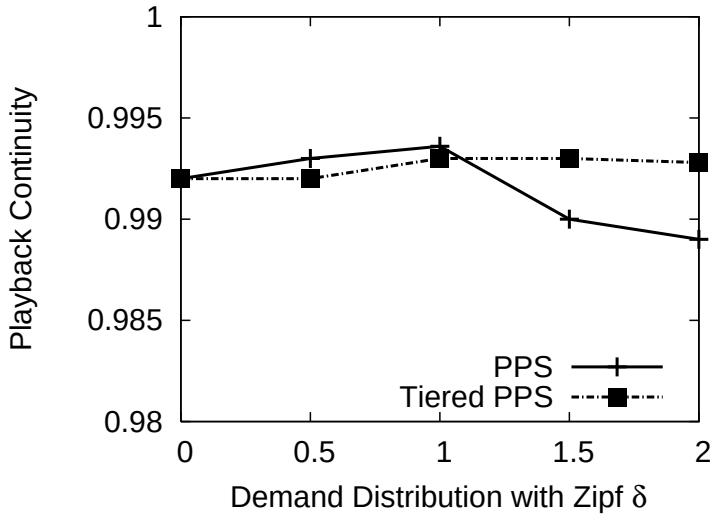

Figure 4.14: The framework helps PPS achieve high and stable performance under various demand distributions.

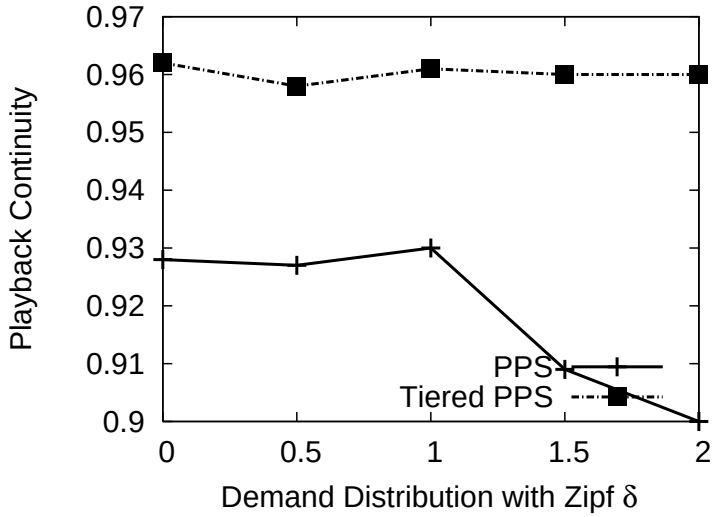

Figure 4.15: The framework enables peers to complete the demand changes within 2 s under various demand distributions.

Figure 4.14 shows that our framework can help PPS achieve high performance even when the demands is unevenly distributed, while PPS before applying our framework suffers low performance when there are few high demand peers. Our framework enables PPS to accommodate peers' heterogeneous demands under various demand distributions. With the tired structure, it is easier for peers inside a cooperative group cooperate with one another. The existence of tire-1 peers makes the system always have the enhancement layer suppliers in all groups.

The values of packet delivery ratio 2 seconds after peers switch to high demands shown in Figure 4.15 demonstrate that our framework helps peers quickly find their desired content and achieve short delays. The improvement is more obvious when the demand distribution is uneven. When most of peers only request the base layer, it is hard for peers to get the enhancement layer after switching to high demands in PPS. With the framework, peers can easily find the suppliers for the enhancement layer within their groups.

Figure 4.16 shows that our framework can reduce the bandwidth cost efficiently.

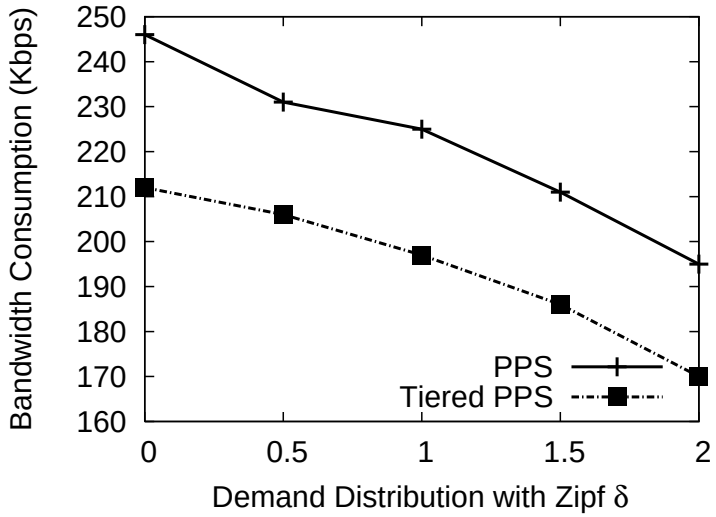

Figure 4.16: The framework considerably reduces the bandwidth consumption under different demand distributions.

When there are lots of peers having the enhancement layer in the system, the improvement on bandwidth consumption decreases little. Because when most of peers request both the base layer and the enhancement layer, the impact of optimal bandwidth allocation becomes less.

4.5 Conclusion and Discussion

In this chapter, we extend our work in Chapter 3 through formulating the problem as a demand-supply problem, and studying the bandwidth allocation optimization. We propose a framework to effectively utilize system bandwidth while still achieving good performance. We formulate the problem with a linear programming model and figure out how peers optimally allocate their bandwidth. Peers are grouped with the tired structure, and cooperate with one another to achieve high bandwidth efficiency, low delay and high video quality. We extensively evaluate the performance of our framework by incorporating it into PPS, an earlier scheme dealing with video

quality demand heterogeneity. Our simulation results demonstrate that optimized algorithms and tiered structure noticeably boost PPS's performance. Apart from PPS, our framework can also be incorporated into other schemes.

Chapter 5

Designing Efficient Time-shifted P2P Streaming Systems with Heterogeneous User Demands on Playback Delay

5.1 Motivation

Time-shifted streaming is an application which allows peers to view live video content in a more convenient way. Peers can view the video segments which have been broadcast before they join the system. The service provider predetermines a time-shifted window to define how much past video a peer can view. For example, when a peer joins the system, he/she can choose to view any video segments within the last 3 hours. Because of this flexibility, time-shifted streaming is very attractive. Although it has been implemented in typical television systems for a while [86], there is not much literature discussing P2P-based time-shifted streaming.

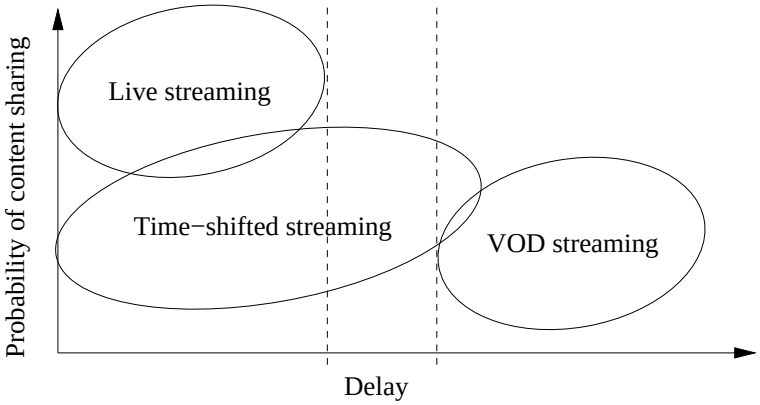

Figure 5.1: Live vs Time-shifted vs VOD streaming.

Time-shifted streaming is similar to live streaming and VOD streaming, but it also has its own specific features. Similar to live streaming, the server in time-shifted streaming continuously broadcasts a live video. But peers in time-shifted streaming can view different video segments of the video. Similar to VOD streaming, time-shifted streaming allows peers to pause and resume, as well as fast-forward and rewind the video. But the video in time-shifted streaming has not yet been recorded and added to the VOD library. Figure 5.1 illustrates their relationship: peers in live streaming view the video when it is broadcast live, and all peers are viewing the same segments. Thus it has the highest probability of sharing segments among peers, and has the shortest playback delay. In VOD streaming, the video is recorded and stored in the VOD library. Peers can view it anytime after that, even several days after it is first broadcast. Thus it has the lowest content sharing probability and the longest playback delay. Time-shifted streaming is an application covering the gap between them. Because time-shifted streaming is different from both live streaming and VOD streaming, existing live streaming and VOD streaming solutions are not suitable for time-shifted streaming.

There are two options to implement P2P-based time-shifted streaming. One is called digital video recorder (DVR) style, in which peers start the application at the

beginning of the live broadcasting, and they use the recording function to save a local copy of the video, so that they can view it later. In the other option, peers can start the application at any time during or even after the live broadcasting to view the video. It does not require recording before their viewing, and we call it non-prerecording (NPR) style. The DVR style can be implemented using existing live streaming solutions with minor changes. But the NPR style is challenging. As peers are viewing different video portions, they might not fetch the video segments in time and glitches may happen. Even if peers can request these segments from the streaming server, the server capacity is limited and the system scalability is affected. In this chapter, we study the design of an NPR style P2P-based time-shifted streaming system with low server bandwidth consumption.

5.2 Time-shifted Streaming Solutions and Challenges

In this section, we firstly study the possible solutions, and investigate time-shifted streaming in most popular of P2P streaming systems. Then we describe the measurement results and discuss the challenges in implementation.

5.2.1 How to Implement Time-shifted Streaming

In P2P based time-shifted streaming, peers are able to view a video asynchronously to the broadcast time. It provides lots of attractive features: 1) pausing the live video and continuing later; 2) watching the past segments of the video and jumping over uninteresting segments until catching up with the live stream. With these features, peers can view the live video with flexibilities like viewing a VOD video. Namely, peers in a time-shifted streaming system are able to access at a time different from

when it is lively broadcast.

In a TV system, time shifting is implemented by recording videos to a storage medium to be viewed at a time more convenient to the consumers [51]. The component used for recording purpose is called digital video recorder (DVR), which allows peers to set up when to start and stop recording. Similarly, the DVR-style option can be used for time-shifted P2P streaming. In the DVR-style option, peers join the system when the live video starts, and record the broadcast content into their local nonvolatile memory. The recording is synchronized with the live broadcasting. When a peer pauses the video, the playing stops but the recording continues. Thus all the broadcast segments are available in the local memory. Peers are able to pause and resume at anytime, and jump to any broadcast segments at anytime. In this method, peers always download the lively broadcast segments no matter whether they pause the stream or not, and which segments they are viewing. Peers' downloading behavior is similar to that in live streaming, so existing live streaming solutions can be used with minor changes.

Different from the DVR-style option, there is another option which does not require the recording before peers join the system, and we call it non prerecording (NPR) option. The NPR-style option is more flexible but also more challenging. In NPR-style P2P streaming, the service provider determines a time-shifted window which indicates how much past video can be viewed. When a peer joins the system, it can access any segments within the window. Please note that the window moves forward along with the live broadcasting. For example, when a peer joins in a system with the time-shifted window of 1 hour at 8 p.m., it is able to view any segments broadcast between 7 p.m. and 8 p.m.. When it joins the system at 9 p.m., it can access segments broadcast between 8 p.m. and 9 p.m.. Peers can view different segments within the time-shifted window. Compared to the DVR-style option, this

	Time-Shifted Streaming	Style
PPStream	No	N/A
PPTV	Yes	DVR
SopCast	No	N/A
TVUPlayer	Yes	DVR
UUSee	Yes	NPR

Table 5.1: Time-shifted deployment in P2P streaming

option is more flexible. But it requires more cooperations among peers, since peers are viewing different segments of the video and it is hard for them to share video segments with one another. Otherwise, it brings too much load to the server.

5.2.2 Time-shifted Streaming in Commercial Systems

We investigate time-shifted streaming in popular commercial P2P streaming systems, and the results are summarized in Table 5.1. SopCast [64] and PPStream [60] do not support time-shifted streaming. PPTV[59] and TVUPlayer [69] already provide DVR-style time-shifted streaming service with which a peer can pause a video after it joins the system. The live broadcast segments are downloaded to peers' local memory, and therefore peers are able to view any of them at any time. UUSee [70] provides NPR-style time-shifted streaming. To the best of our knowledge, only UUSee allows peers to access the video segments which have been broadcast about 40 minutes before their joining times.

To understand how commercial P2P time-shifted streaming systems work, we measure three systems. 1) PPTV selected as a representative of DVR-style time-shifted streaming, 2) PPStream selected as a representative of no time-shifted streaming, and 3) UUSee selected a representative of NPR-style time-shifted streaming. In PPTV, we pause and resume the video periodically. We observe that the download rate is almost constant, as peers download the live broadcast segments at the streaming rate

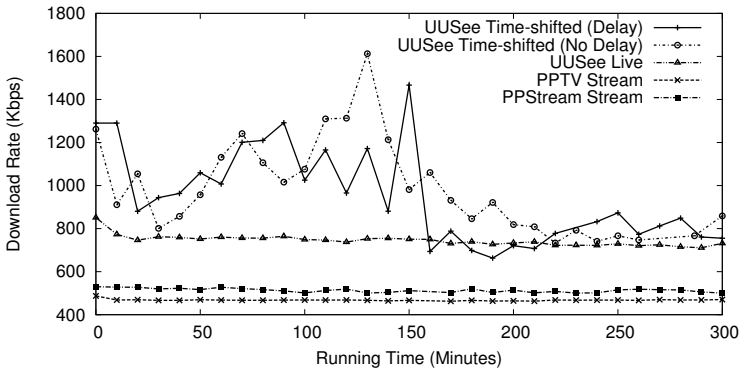

Figure 5.2: Peers in time-shifted streaming of UUSee also download the segments which are not needed for immediate viewing.

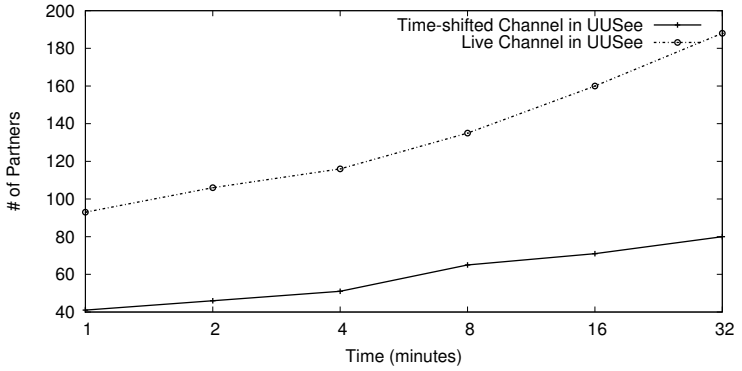

Figure 5.3: Peers in UUSee time-shifted streaming have much less partners than peers in live streaming.

of the video like a digital video recorder. In PPStream, the peer's download rate is not affected by pausing and resuming the video either. PPStream does not support DVR-style time-shifted service. It automatically resume after pausing for a short period when its buffer is fulfilled. A minor change at the client end can make PPStream support the same feature as PPLive: peers periodically move the content in the buffer to their local disks, so the memory space can be used for new video segments. In UUSee there are two independent channels for the same live broadcasting. For example, there is a channel for CCTV-6 which only provides live streaming service, and there is another channel for it providing time-shifted

service in a new channel does not affect the corresponding live channel which can still serve all peers in the same way as before introducing the time-shifted one. However, this method has its disadvantages, and we will discuss them later.

We not only measure the time-shifted channel but also consider the live channel as a reference in UUSee. In Figure 5.2, we can see that peers in time-shifted streaming have higher download rates than peers in live streaming till they join for more than two and half hours. Theoretically, the streaming rates of the same video should be the same or at least close in time-shifted streaming and in live streaming. Because peers in time-shifted streaming prefetch segments which may not be required by themselves right now, the download rate in a time-shifted streaming is higher than the video streaming rate. Please note that peers can "prefetch" both the future segments and the past segments. This can be proved by the fact that peers are able to jump forward or backward to another segment smoothly after they join the system long enough.

Peers stop prefetching when there are sufficient segment copies in the system. This is why finally the download rate in time-shifted streaming gets close to the rate in live streaming. For the time-shifted channel, we have two sets of data, one of which is the download rate of peers which view the video synchronously with the live broadcasting, while the other is the download rate of peers which view the video with the maximum time-shifted delay. There is a small difference between peers with delay and without delay on when the download rate drops close to the download rate in live streaming. What peers are viewing may affect their downloading behavior, so their download rates are slightly different.

We also notice that UUSee provides time-shifted streaming service and live streaming service with two separate channels. We check the distinct source IPs of packets received by two peers watching CCTV-6 at the same time but one in the live streaming channel and another in the corresponding time-shifted streaming channel, respec-

tively. As shown in Figure 5.3, the peer in the time-shifted streaming channel has less partners compared to the peer in the live streaming channel. We also find that the video playback is not very smooth in the time-shifted streaming channel. This is reasonable as most people view the live broadcast video and they join the live streaming channel, so peers in the time-shifted streaming channel are less than peers in the live streaming channel. With the two separate channel method, none of existing live streaming channels is affected after introducing time-shifted streaming service into UUSee. But it may cause problems: when a peer in the time-shifted streaming channel leaves, its partners may not easily find a substitute. Thus their video has glitches. We claim that we should use one channel to provide both time-shifted and live streaming services, and further discuss and analysis are provided in the following section.

The measurement results show that prefetching is used to implement NPR-style time-shifted streaming. Prefetching is critical as peers have different interests and their segment sharing opportunities are very low. Without prefetching, peers can hardly fetch segments from other peers. Further, prefetching sometimes also benefits peers directly. For example, if peers jump to past segments which have been prefetched by themselves, they can view the segments without any delay. But UUSee uses its proprietary protocol, and the details are still obscure. In this chapter, we study efficient prefetching in NPR-style time-shifted streaming, and try to shed light on the design and implementation of it.

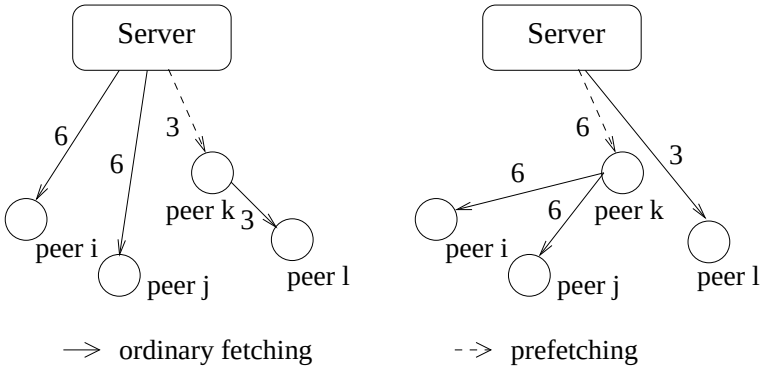

Figure 5.4: A good prefetching strategy reduces server load without impairing video quality.

5.3 Prefetching in NPR-style Time-shifted Streaming

Which segments peers prefetch is critical in NPR-style time-shifted streaming, and a good prefetching strategy results in high quality service and low server cost. Let us consider the case in Figure 5.4: peers i and j are requesting segment 6 and peer l is requesting segment 3. Peer k has 2 units of upload capacity. On the left side of Figure 5.4, peer k prefetches segment 3, then forwards it to peer l. The other two peers cannot get segment 6 from other peers, and directly request it from the server. In contrast, if peer k prefetches segment 6 as shown on the right side, it can fully utilize its upload bandwidth to forward segment 6 to both peers i and j. Peer l may be able to fetch segment 3 from other peers. Even if peer l fetches segment 3 from the server as shown in the figure, the server bandwidth consumption is reduced by 1 unit. Peers can prefetch any available segments, and they do not need to prefetch segments in order. Figure 5.5 shows that peer k prefetches segments 4 and 6 and skips segment 5. Because it is going to view segment 4 and segment 6 is the one which is highly demanded by other peers.

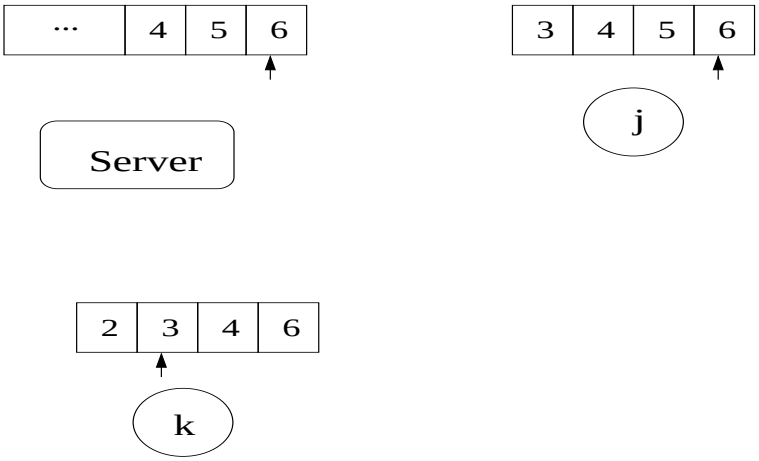

Figure 5.5: Snapshot of local cache at peers

5.3.1 System Model

A time-shifted system consists of a server S and $|N(t)|$ peers with p_i representing the i-th peer, U_i and D_i representing its upload and download bandwidth, respectively. The system allows peers to view past video with a maximum shifted time of T. The server continuously generates and injects new segments into the system, and the available segments are within the time-shifted window of $[t-T, t]$ where t is the current time. Let t_i denote p_i's viewing time at t, and we have $t - T \leq t_i \leq t$. Obviously, we can consider a live streaming system as a special case with $T = 0$.

We assume that peers have sufficient storage to store all segments in the time-shifted window. As the window is usually a limited size covering hours of the video, the assumption is reasonable and we do not need to consider segment deletion in case of peers running of the storage. For example, a program such as a sitcom episode or a sports game lasts at most hours, and the time-shifted window covers hours of the video. Peers joining the system when the program is being broadcast are able to view any segment of the program from the beginning to the current time. The $[t-T, t]$ window moves forward along with t, and we use $M(t)$ to denote the set of segments provided by the system at time t. For example, in a system with a time-shifted

window of 1 hour long, peers are able to view any video segments broadcast between 7 pm and 8 pm at 8 pm. They can view any video segments broadcast between 8 pm and 9 pm at 9 pm.

Let us consider peers' asynchronous viewing activities and their local memory status in the time-shifted streaming. Any segments within the $[t-T,t]$ window are available at the server, but not all of them are necessarily possessed by a peer. We use $I_i^j(t)$ to denote if p_i has segment j at t. Let a 0-1 variable $V_i^j(t)$ represent whether p_i immediately views segment j at t. Peers can view any segments within $[t-T,t]$, which is determined by their subjective choices. They fetch segments according to these $V_i^j(t)$'s and their local memory status. Namely, they must fetch the segments which they will view, but are not yet in their local memory. Peers also prefetch some segments to improve segment availability at peers, and we use $P_i^j(t)$ to show if p_i prefetches segment j at t. The prefetched segments may be viewed in the future or just be downloaded for forwarding to other peers. The value of $P_i^j(t)$ does not affect peers' current viewing experience as the segment is not emergent, but peers may view the segment themselves or forward the segments to other peers in the following time slots. $V_i^j(t)$ and $I_i^j(t)$ are known in our context, and $P_i^j(t)$ is the variable that is optimized based on the values of $V_i^j(t)$ and $I_i^j(t)$.

5.3.2 Problem Formulation

Our primary objective in this chapter is to design an efficient prefetching method in peer-assisted time-shifted streaming, so that peers are able to get most segments before their playback deadlines, and most of the segments are fetched from other peers instead of the streaming server.

In a time-shifted streaming system, peers are able to view any segments within the window $[t-T,t]$. The segment requests which could not be served by other peers are

finally sent to the server. Segment prefetching improves the supply of the segments, but on the other hand, segment prefetching also consumes bandwidth and it should not affect users' viewing experience. Then the problem can be formulated as:

$$min \sum_{j \in M(t)} \Big(\sum_{i \in N(t)} V_i^j(t+1) * (1 - I_i^j(t+1)) - \sum_{i \in N(t)} I_i^j(t+1) U_i \Big)$$

$$s.t. \sum_{i \in N(t)} (V_i^j(t) * (1 - I_i^j(t)) + P_i^j(t)) \leq \sum_{i \in N(t)} I_i^j(t) U_i, \ \forall j \tag{5.1}$$

$$\sum_{j \in M(t)} \sum_{i \in N(t)} (V_i^j(t) * (1 - I_i^j(t)) + P_i^j(t)) \leq \sum_{j \in M(t)} \sum_{i \in N(t)} I_i^j(t) U_i, \ \forall i, j \tag{5.2}$$

$$\sum_{j \in M(t)} (V_i^j(t) * (1 - I_i^j(t)) + P_i^j(t)) \leq D_i, \ \forall i \tag{5.3}$$

$$I_i^j(t+1) = I_i^j(t) + V_i^j(t) * (1 - I_i^j(t)) + P_i^j(t) \tag{5.4}$$

$$P_i^j(t) = 0, \ if \ I_i^j(t) = 1 \tag{5.5}$$

In this formulation, $P_i^j(t)$'s determine the prefetching solution. They are binary optimization variables. The objective is to find the optimal prefetching variables $P_i^j(t)$'s to minimize the requests sent to the server.

Inequalities (5.1) and (5.2) are constraints ensuring that prefetching should not affect peer viewing experience. Inequality (5.3) is peer download bandwidth constraint. Eq. (5.4) is the updating equation for peer's local status. Eq. (5.5) is used to remove the duplicated fetches. Both the prefetched segments and the regular fetched

segments are stored in peers' local storage and peers can forward them to other peers in the following time slots.

According to the objective formula, we have two ways to minimize the number of outstanding requests: decreasing the requests and increasing the supply. What peers are going to view is completely determined by themselves, and thus the optimization objective can be expressed as a maximizing problem after expanding the minimization formula:

$$max \sum_{j \in M(t)} \Big(\sum_{i \in N(t)} V_i^j(t+1) * I_i^j(t+1) + \sum_{i \in N(t)} I_i^j(t+1) U_i \Big) \tag{5.6}$$

We update Formula (5.6) using Eq. (5.4). As peers' local status ($I_i^j(t)$), what peers are viewing ($V_i^j(t)$) and what they will view in the next time slot ($V_i^j(t+1)$) are known in our context, the terms which consist of these variables are fixed. We have the equivalent problem after updating the optimization problem by removing these terms:

$$max \sum_{j \in M(t)} \Big(\sum_{i \in N(t)} V_i^j(t+1) * P_i^j(t) + \sum_{i \in N(t)} P_i^j(t) U_i \Big) \tag{5.7}$$

Based on Formula (5.7), we know that prefetching is affected by what peers are going to view and their upload capacities.

5.3.3 Heuristic Distributed Algorithm

The whole system information is required to calculate the optimal prefetching solution. For example, we need to know what peers are viewing, which segments exist in

their local storage, how much their available upload bandwidth is, and so on. Though the optimal prefetching gives the lower bound of the server bandwidth consumption, the objective in this chapter is to design a practical solution that can be implemented in a decentralized way.

We propose our heuristic distributed algorithm by decomposing the global optimization problem into multiple small optimization problems running on individual peers. For each single peer, it collects the local information from its neighbors, and then determines which segments to prefetch. Similar to the global optimization problem, we have the local optimization objective: a peer prefetchs segments so that it can forward the segments required by their neighbors as many times as possible. Thus, we have the localized format of Formula (5.7):

$$max \sum_{j \in M(t)} \Big(\sum_{k \in NBR_i(t)} V_k^j(t+1) * P_i^j(t) + P_i^j(t) U_i^j(t) \Big) \tag{5.8}$$

Where $NBR_i(t)$ is the neighbor set of peer i, and $U_i^j(t)$ is peer i's upload bandwidth allocated for segment j.

Similarly, the constraints also have the corresponding local format by replacing N_t with $NBR_i(t)$ and U_i with the bandwidth allocated by peer i.

Usually, it is hard to know the accurate bandwidth that a peer allocates to a specific segment in a distributed system. Further, a peer is both a request sender and a request receiver. Therefore we divide the algorithm into the request generation sub-algorithm and the bandwidth allocation sub-algorithm. A peer uses the request generation sub-algorithm to figure out the most demanding segments of its neighbor peers, and sends prefetching requests for these segments first. It uses the bandwidth allocation sub-algorithm to determine which received requests should be served, to increase the supply of the most demanding segments. Both of them have the same

objective in Formula (5.8). In the request generation sub-algorithm, a peer not only finds the most demanding segments, but also determines how much bandwidth it can allocate for these segments. The bandwidth is used to forward the segments to other peers after the peer receives them.

In our simulation, peer i periodically calculates its residual bandwidth using its upload bandwidth to subtract the consumed bandwidth in the previous period. It determines which segments to prefetch based on the demand and supply of the segments from its neighbors. Then it calculates bandwidth that it can allocate to these requested segments, which is called the committed bandwidth. The segment requests are sent with this committed bandwidth information. If the peer does not have enough residual bandwidth to compensate the bandwidth that it consumes to prefetch a segment, it stops prefetching. When peers receive the requests, they always serve the requests which are for the rarest segments and/or the ones with higher committed bandwidth first. Tables (5.2) and (5.3) are the pseudo code for request generation and bandwidth allocation sub-algorithms on a single peer, respectively.

We can see that no global knowledge or synchronization are required in these algorithms, and the prefetching solution of a peer does not depend on the solution of another peer. Thus, the algorithm is scalable and practical.

5.3.4 Two Channels or One Channel?

In UUSee, live streaming and time-shifted streaming are provided in two separate channels, which is not necessary. Actually we can use a single channel providing both live streaming and time-shifted streaming. Further using two separate channels may cause more direct requests to the server.

As we focus on a specific time, we can ignore the time variable. Let us use N_l to represent the peers which use the live streaming in the whole peer set of N. With

```
if p_i is going to view segment j in the next time slot then
  Checking if it is in the local memory I_i^j = 1
  if not in local memory then
    p_i requests segment j
  end if
end if
for any other segments within the shifted window do
  if p_i does not have them then
    p_i checks its neighbors' memory status to find the rarest segment
    calculates its residual bandwidth
    if residual bandwidth > bandwidth consumed by downloading
    a segment then
      requests it with committed bandwidth
      min(residual bandwidth, 2*bandwidth consumed by segment
      prefetching)
    else
      break
    end if
  end if
end for
```

Table 5.2: Pseudo code: which segments to request.

```
for all received requests do
  if it has residual bandwidth then
    serves the requests for segments which sending peers need to
    view in the next time slot
    if still has residual bandwidth then
      finds the requests for the rarest segment in its neighborhood
      serves the one with highest committed bandwidth
      if the committed bandwidth is equal, randomly choose one of
      the requests
    end if
  end if
end for
```

Table 5.3: Pseudo code: which requests to serve.

UUSee's method, these peers join the live streaming channel and the peers in $N - N_l$ join the corresponding time-shifted channel. In the live streaming channel, all peers are synchronized to view the same video. The number of requests sent to the server is determined by the streaming rate R and peers' aggregated upload bandwidth. We

get the number of requests directly sent to the server in the live streaming channel as

$$max(|N_l| * R - \sum_{i \in N_l} U_i, 0) \tag{5.9}$$

While for the peers in corresponding time-shifted channel, the expected requests received by the server is

$$max(\sum_{j \in M} (\sum_{i \notin N_l} V_i^j * (1 - I_i^j) - \sum_{i \notin N_l} I_i^j U_i), 0) \tag{5.10}$$

If we use a single channel method, the total number of requests sent to the server can be calculated as

$$\begin{aligned} & max(\sum_{j \in M} (\sum_{i \in N} V_i^j * (1 - I_i^j) - \sum_{i \in N} I_i^j U_i), 0) \\ & = max(|N_l| * R - \sum_{j \in M} \sum_{i \in N_l} I_i^j U_i \\ & \quad + \sum_{j \in M} (\sum_{i \notin N_l} V_i^j * (1 - I_i^j) - \sum_{i \notin N_l} I_i^j U_i, 0) \end{aligned} \tag{5.11}$$

Obviously, peers which use live streaming have the segments within the window M. Namely,

$$\sum_{j \in M} \sum_{i \in N_l} I_i^j U_i = \sum_{i \in N_l} (\sum_{j \in M} I_i^j) U_i \geq \sum_{i \in N_l} U_i \tag{5.12}$$

So we have (5.11) $\leq$ (5.9) + (5.10). The single channel method generates less requests directly sent to the server compared to using two separate channels in UUSee. This is easy to understand: dividing peers into two separate channels disables some possible peer cooperation. The residual bandwidth in one channel cannot be used to help peers in the other channel. Further, with the time-shifted feature, peers losing

connections because of their own network or their partners' departures can still view the video without missing any content. This is another advantage of the single channel method.

5.4 Performance Evaluation

We developed a simulator to evaluate the performance of our method (referred to as NPR), comparing with the following four methods. (1) Initial Playback Position Caching (IPP) proposed in [13]: a peer caches the content starting from the peers' initial playback point; (2) Live Stream Position Caching (LSP) proposed in [13]: a peer caches the content starting from the system live streaming point when the peer joins the system [13]. For example, a peer joins the system at 8 pm, and chooses to view the video content at 7 pm. It caches the content starting from 7 pm in IPP, whereas it caches the content starting from 8 pm in LSP. (3) UUSee style method (referred to as NPR2) which has two channels for the same program: one for live streaming and the other for time shifted streaming. Please note that NPR and NPR2 are the same when there is only live streaming or only time shifted streaming. NPR2 is used as a reference to show that whether we need to separate peers into two channels providing live streaming and time shifted streaming, respectively. (4) The optimal method (referred to as Optimal) which is a centralized algorithm solving Problem (1) and is used as a reference method. There are some other related methods [51], but they are tree-based. Since we implement a mesh-based overlay which is used in most commercial systems, and applying these methods in a mesh-based overlay makes them quite different from the original ones. Therefore they are not included in the simulation.

In our simulation, there are 1000 peers by default, and they have heterogeneous

bandwidth. To simplify the description, we only have two types of bandwidth: 20% of peers called high bandwidth peers have the upload bandwidth of 3 segments per time slot and download bandwidth of 10 segments per time slot. 80% of peers called low bandwidth peers have the upload bandwidth of 1 segment per time slot and download bandwidth of 5 segments per time slot. The settings reflect that most of peers do not have high bandwidth and the asymmetric upload/download bandwidth in the Internet. In addition, we also simulate other settings with different bandwidth distributions, and get the similar results. In all experiments, the streaming server constantly generates 1 segment per time slot, and a maximum of 300 past segments are available by default. When peers cannot fetch segments that they will view in the next time slot from other peers, they send their requests to the server. In order to focus on the impact of these methods on the server bandwidth cost, we assume that the server has unlimited upload bandwidth. We define the server bandwidth cost to be the percentage of the server bandwidth consumption in the whole system bandwidth consumption. For example, if 50% segments streamed to peers are provided by the server, the sever bandwidth cost is 50%.

We implement the following 4 sets of experiments: 1) peers dynamically join and leave the system, and randomly choose any segments between $t-T$ and t to view, 2) different available shifted period Ts, 3) different viewing distributions, 4) peers jump backward and forward.

5.4.1 Impact of Peer Churn

In this group of simulations, we study whether these methods are robust to peer churn. Peers dynamically join the system as a Poisson process with an average arrival rate of 10 peers per time slot and their online durations follow an exponential distribution with a mean of 500 time slots. A peer randomly chooses a playback time between $t-T$

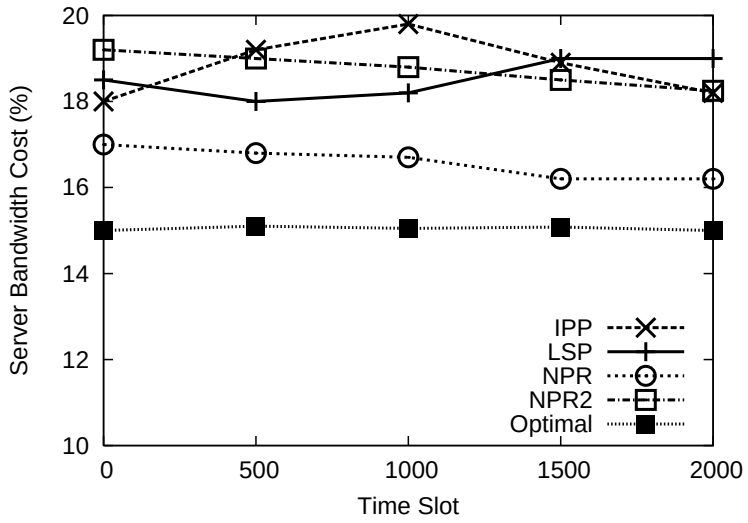

Figure 5.6: NPR considerably reduces the server bandwidth cost through prefetching.

and t according to the uniform distribution. Figure 5.6 shows their performance: our NPR method consumes less server bandwidth than IPP, LSP and NPR2. Because in our method, peers utilize the spare bandwidth to fetch segments which are not required by themselves but highly demanded in the system, and this improves the segment availability at peers. NPR2 prevents peers in a channel to help peers in another channel, and thus its server bandwidth cost is higher than NPR. However, peers have limited neighbors and their decisions are based on their local information in our simulation. Thus, there is a performance gap between our method and the optimal one. It is more obvious when the system changes more dynamically, as shown in the following subsection.

As we discussed before, prefetching implies more bandwidth consumption because it downloads segments which are not required immediately by a peer. In Figure 5.7, we present the extra bandwidth consumption introduced by each method. We define the extra bandwidth cost as the percentage of extra consumed bandwidth compared to the video streaming rate. For example, if the video streaming rate is 1 segment/slot, and the actual average download rate is 1.1 segments/slot, we have 10% extra bandwidth

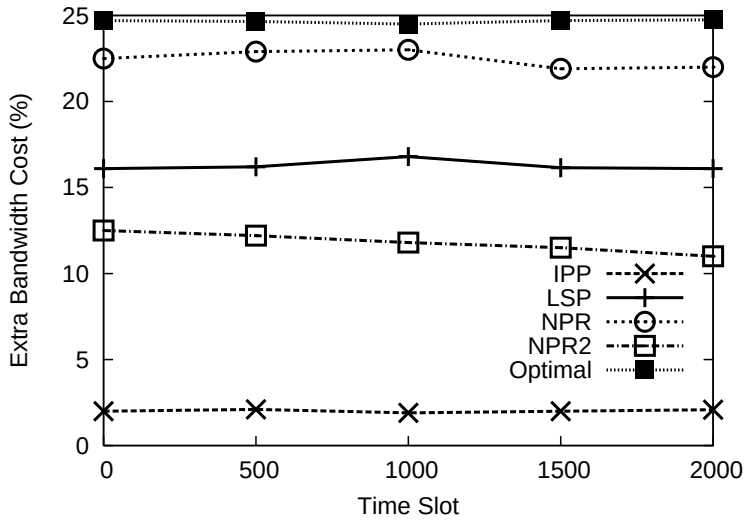

Figure 5.7: Prefetching in NPR also consumes extra peer bandwidth.

cost.

Our NPR method consumes more extra bandwidth than LSP. Because in our method, peers can prefetch any available segments. While peers in LSP only fetch the current segments and the segments which they are going to view. Peers in IPP do not prefetch any segments, and there is some control overhead. Therefore, the extra bandwidth is close to 0. The extra bandwidth cost in other simulations has the similar results and we do not present them in this chapter.

5.4.2 Impact of the Maximum Shifted Time

In this group of simulations, we investigate the impact of the maximum shifted time on the server bandwidth cost through changing the value of T from 150 to 600 time slots. As shown in Figure 5.8, the longer the maximum shifted time is, the higher the server bandwidth cost is for all the methods. This is because peers have more playback points to choose with a longer maximum shifted time. Therefore, the segment sharing becomes more challenging. Since peers are able to identify which segments are more scare using our prefetching method, the server cost increases more slowly with NPR

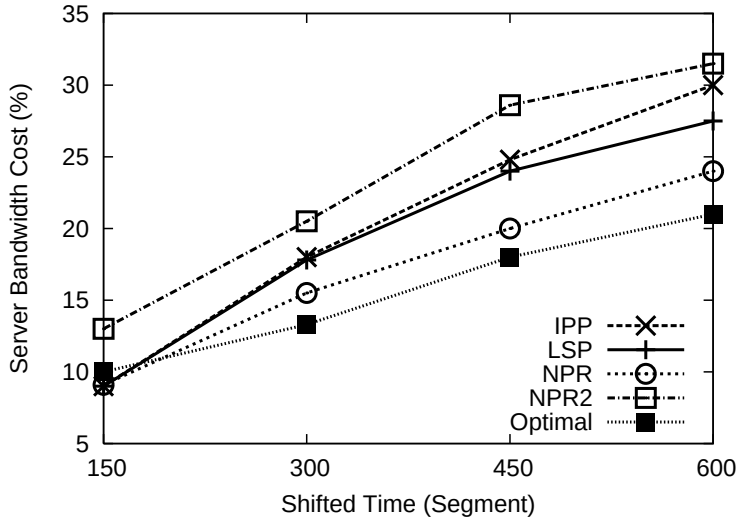

Figure 5.8: The longer the maximum shifted time, the higher the bandwidth cost. Prefetching helps NPR alleviate the impact.

than with IPP and LSP.

5.4.3 Impact of the Initial Playback Point Distribution

In a time-shifted streaming, peer viewing time may not be uniformly distributed between $t - T$ and t. For example, the initial playback points are more likely to be close to the beginning of a video than to the end of the video. We equally divide the video in $[t-T, t]$ into three sections, which represent the beginning, the middle and the end, respectively. We conduct the experiment in which peers' initial playback points are in one of three sections. The numbers of peers in these three sections follow Zipf's law which is an empirical law formulated using mathematical statistics [1]. The Zipf parameter controls the number of peers viewing each section. Namely, if the Zipf parameter is small, most of peers are viewing the end section; whereas most of peers are viewing the beginning section when the parameter is large. A peer firstly determines which section it wants to view, then it randomly chooses the viewing time in the corresponding section. For example, if a peer chooses to view the beginning

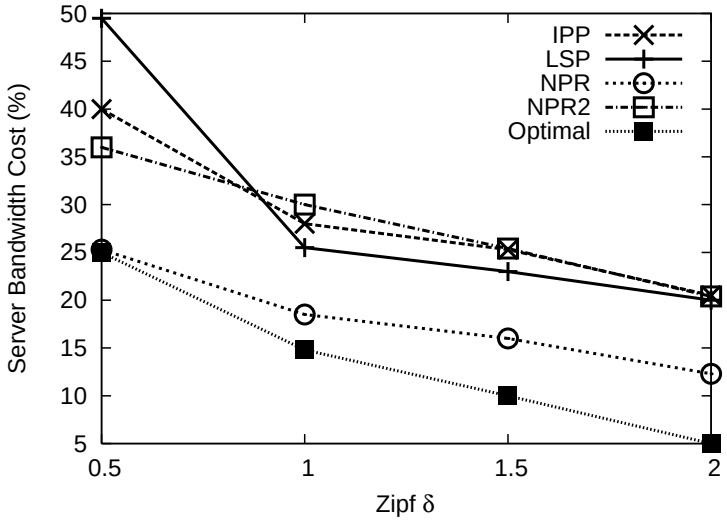

Figure 5.9: NPR pretching method achieves low server bandwidth cost with different initial playback point distributions.

section, its viewing time can be any value in $[t - T, t - \frac{2T}{3}]$. Figure 5.9 indicates that the more peers view the beginning section, the less the server bandwidth cost is. This is because usually the segments in the beginning section are cached more than the recent segments. In our NPR method, peers determine which segments to fetch and cache based on the segment availability and demand, so it always consumes less server bandwidth.

5.4.4 Impact of Dynamic Viewing Activities

We also show the impact of peer jumping backward and forward within the shifted time window. Peers jump to view other video segments after they join the system. The intervals between a peer's two consecutive jumps are uniformly distributed between 0-500 time slots. In Figure 5.10, we can see that the server bandwidth cost is very high for all the methods. But our method steadily decreases the server bandwidth cost after 500-600 time slots. As our NPR method forces peers to aggressively fetch segments using peers' spare bandwidth, they maintain a considerable number

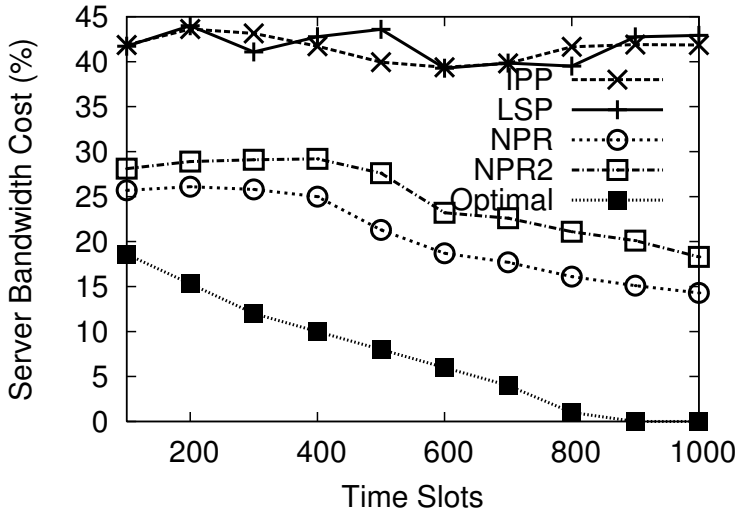

Figure 5.10: Peers randomly jump backward/forward, average 10 times per peer in this experiment. NPR prefetching method reduces the server cost over time.

of segments in its local memory. Thus peers are able to view the segments from their local memory or fetch them from other peers.

5.5 Conclusion and Discussion

In this chapter, we study the design of NPR-style time-shifted streaming with low server cost using segment prefetching. We first formulate it as a problem of minimizing the server bandwidth cost. Then we propose our distributed heuristic algorithm called NPR method, with which each peer determines its segment scheduling. The simulation results show that our NPR method can considerably reduce the server bandwidth cost. Although the extra peers' bandwidth consumed by prefetching in our NPR method does not impair users satisfaction, it is worth to figure out the lower bound of the extra bandwidth. In the NPR method, peers aggressively prefetch segments whenever there is idle bandwidth. However, it is not necessary if the system is relatively stable and peers do not frequently jump backward/forward. If we can

predict what peers are going to view in following time slots and do not prefetch the segments are rarely viewed, the overall bandwidth cost can be reduced. We will investigate segment prediction algorithms in the future.

Chapter 6

Designing Efficient Real-time P2P Streaming Systems with Heterogeneous User Demands on Playback Delay

6.1 Motivation

Peer-to-Peer (P2P) live streaming has been widely used for large-scale video distribution [70]. However, the playback delay, which is the difference between the time when the video is originally streamed out from the server and the time when a peer receives it, increases with the increase in the system size. Some peers would like to view delay-sensitive videos (e.g., breaking news, live sports etc.) within a bounded delay even if they need to pay for the bounded delay. We call them *subscribers* to differentiate them from the ordinary peers and we define the *subscriber bounded delay (SBD)* problem as streaming the video to subscribers within the delay bound.

Providing subscribers with delay-bounded service is important since it can improve user experience and can help service providers gain extra profits. However, current research studies mainly focus on minimizing overall delay in a P2P streaming system, which cannot be directly used for providing delay-bounded service to subscribers. Because these delay minimization solutions do not differentiate between subscribers and ordinary peers, the delay requirement of subscribes may be violated if they are placed far away in the streaming overlay from the streaming server. To view the video within the bounded delay, these subscribers need to establish new connections, e.g., by directly requesting the video from the server. Such a mechanism impairs the system scalability.

Note that it is impossible to guarantee the bounded delay of every subscriber at any time. There are two major reasons. First, peers in a P2P streaming system may dynamically join or leave the system. As a result, the average bandwidth per peer in the system is dynamic. Second, all peers and the streaming server have limited bandwidth, and thus they have a limited number of neighbors in the streaming overlay. As a result, the average number of hops between the streaming server and a peer in the system increases as the total number of peers increases. Even if we use admission control, we still cannot absolutely guarantee the bounded-delay of subscribers due to peer dynamics. Therefore, in this chapter, we aim to serve as many subscribers as possible with bounded delay service, and serve the remaining subscribers and ordinary peers with best-effort delay service.

In this chapter, we first formulate the SBD problem as a decision problem, and prove that it is NP-Complete by reducing the bounded Steiner tree problem [61] to it. We propose the concept of peer's fanout capacity which is the total amount of upload bandwidth that it can allocate to stream the video to subscribers. Then we propose a centralized algorithm and a distributed implementation called high fanout

promotion (HFP) algorithm to efficiently serve the as many subscribers as possible.

6.2 Problem Formulation and Design Overview

In this section, we first formulate the subscriber bounded delay problem and show that it is NP-complete. Then we propose a centralized heuristic design which helps the system to serve subscribers with the bounded delay video as many as possible.

6.2.1 Problem Formulation

We consider a single channel P2P live streaming system, and model it as a graph $G = (V, E)$, where V is the set of all participating peers including the server, and E is the connection set among peers. Peer i has an upload bandwidth of U_i. A streaming server S_0 with upload capacity U_0 serves all peers in this channel. Note that we use a single streaming server to simplify the description of our model. Our model has no limitation on the number of streaming servers. The video streaming rate is R. Based on studies in [7], the bandwidth bottleneck occurs at network edges. Therefore, a peer's upload bandwidth determines how many other peers it can serve.

We denote the subscriber set with V_S and $V_S \subseteq V$. All subscribers prefer to receive the video within the delay bound T. For any edge $< i, j > \in E$, we use $d_{i,j}$ to represent the link delay between peer i and peer j. Let D_i represent the playback delay at peer i. If peer i receives video segments from peer j, the link delay $d_{j,i}$ is accumulated at peer i. Let P_i be peer i's upstream peer set, which consists of one or more peers. The playback delay at peer i can be presented as:

$$D_i = max_{j \in P(i)}(D_j + d_{j,i}) \tag{6.1}$$

We use $r_{j,i}$ to represent the rate at which peer j forwards the video segments to peer i. Peer i has $\sum_{j \in P_i} r_{j,i} = R$.

The *subscriber bounded delay problem (SBD problem)* can be described as:

Given U_0, identify $r_{i,j}$, $\forall i, j \in V$ such that the following constraints are satisfied:

$$\sum_{j \in P_i} r_{j,i} = R, \ \forall \ i \in V, \tag{6.2}$$

$$D_i \leq T, \ \forall \ i \in V_S, \tag{6.3}$$

$$\sum_{j \in V} r_{i,j} \leq U_i, \ \forall \ i \in V \tag{6.4}$$

Eq. (6.2) ensures that all peers are fully served with the smooth video. Inequality (6.3) is the subscribers' bounded delay constraint, and Inequality (6.4) is the bandwidth constraint.

Theorem 1. *The SBD problem is NP-Complete.*

Proof. It is easy to see that the SBD problem is in NP: given all the peer-to-peer delay information and a solution, we can determine the playback delay of all peers. Then we verify the solution by comparing the subscribers' playback delay with the delay bound T. This can be done in polynomial time.

The NP-complete Steiner tree decision problem [61] can be reduced to the SBD problem in polynomial time. Given an edge-weighted graph $G = (V, E, W)$ in which V is the vertex set, E is the edge set, and W is the corresponding weight set, and a subset $V_S \subseteq V$, a k-value Steiner tree decision problem is to determine whether a Steiner tree of total weight at most k exists. We construct the SBD problem on the overlay $G' = (V', E', W')$, with a subset V'_S and the delay bound $T = k$. Peer set V' and subscriber set V'_S are identical to V and V_S in the Steiner tree decision problem, respectively. For every edge $e \in E$, we have a corresponding edge $e' \in E'$ with the

delay $w' = w$. For any pair of peers in V'_S, e.g., (i, j), we add an edge $< i, j >$ to E' with an infinite delay if $< i, j >$ is not yet in E'.

Next we update the edge delays: starting from a set O with only one element - server 0, we randomly pick up a peer i in V'_S, and find the path with the lowest delay between it and any other peers in O. Then update the edges from peer i to all peers in O with this lowest delay. After this, we add peer i and all peers on the path but not yet in O into O. Finally, we choose another remaining peer in V'_S, and repeat the aforementioned steps till all peers in V'_S have been added to O. This can be done in polynomial time. After the reduction, we have a path connecting all subscribers with the minimum delay which is equal to the minimum weight of the Steiner tree in the decision Steiner tree problem.

For any peer $i \in V'$ including the server, we set its upload bandwidth equal to the video streaming rate, i.e. $U_i = R$. Namely, peer i can just serve one other peer, and every peer has exactly one parent and one child. The optimal solution of minimizing the maximum delay at subscribers is starting from the server 0 and go through all peers in V'_S. The maximum subscriber playback delay is the total path delay from the server 0 to the last subscriber. Whenever we have an instance for k-value Steiner tree in G, we can find a corresponding path starting from the server and covering all peers in V'_S with delay less than or equal to T in G'. And vice versa.

Thus, SBD is NP-Complete. □

6.2.2 Centralized Heuristic Design

Since the SBD problem is NP-Complete, we are not able to find a polynomial algorithm to determine if there exists a solution for all subscribers to view the delay-bounded video. In this section, we develop a heuristic to serve as many subscribers as possible with bounded delay. In other words, this heuristic minimizes the number of

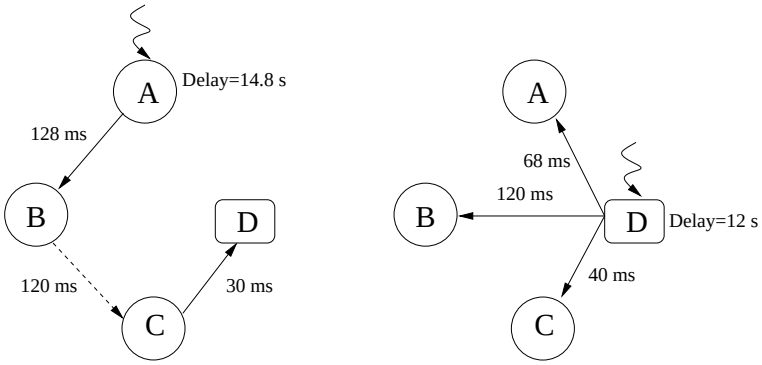

Figure 6.1: In some cases, placing non-subscriber peers close to the server can help reducing the subscribers' playback delays.

delay bound violations at the subscribers. Further, the delay-bounded service should be stable. That is, when the system cannot support any more subscribers, it should decline any newly incoming subscribers' requests for the delay-bounded service. These new subscribers receive the best-effort delay service.

According to Eq. (6.1), the playback delay is determined by the maximum number of overlay hops traversed by the video, and the delay on each link. It is straightforward to construct a streaming mesh in which we place the subscribers close to the server. Because this method makes the local optimal choice, we call it the greedy method.

However, the greedy method consumes lots of server bandwidth in some cases. Let us consider the example in Figure (6.1) which shows a part of the overlay and the playback delay bound is 15 seconds. Peers A, B and C are subscriber peers, and peer D is an ordinary peer. Peer D has higher upload bandwidth compared to the subscriber peers, and it is geographically close to the server. On the left side, the greedy method places subscriber peers close to the server and the ordinary peers get the video from the peripheral subscriber peers. The subscriber peers have limited upload bandwidth, so the video traverses many hops to reach the peripheral subscriber peers (e.g., subscriber peer C). Thus, the playback delay at C violates the delay bound. In this case, peer C needs to request the video directly from the

server. The right side shows an alternative which inserts the ordinary peer D between the subscriber peers and the server, such that it streams the video to all the three subscriber peers without delay violation, and no extra server bandwidth is required.

A sophisticated heuristic should construct a mesh where subscriber peers have playback delays within the delay bound and only a few peers directly connect with the server. To achieve this, some ordinary peers are inserted in front of subscriber peers as shown in the aforementioned example. Here we define a peer's fanout capacity to help explain our design. Usually, it is defined as how many other peers it can serve. In this chapter, the concept is a little different. We define the peer's fanout capacity based on the peer's upload bandwidth and the playback delay. Let $F(i)$ be the fanout capacity of peer i, which is defined as how much service it can provide to subscriber peers within the bounded delay, i.e.,

$$F(i) = \sum \phi(j) \qquad \forall\ j \in V_S \tag{6.5}$$

where

$$\phi(j) = \begin{cases} r_{i,j} & if\ D_i + d_{i,j} \leq T \\ 0 & otherwise \end{cases}, \sum \phi_j \leq U_i$$

To reduce the direct connections to the server, peers with high fanout capacities are placed close to the server. Our heuristic runs as as follows: peers are sorted based on their fanout capacities. Then we push the peers one by one into the system in descending order. Every time we push a peer, we update the remaining peers' fanout capacities: if the pushed peer is a subscriber, it is removed from V_S. It is possible that the system cannot serve all subscribers with delay-bounded service. The heuristic then treats the unsatisfied subscribers as ordinary peers since the system

cannot serve any more subscribers with the delay-bounded video. Further, it needs to provide existing subscribers with delay bounded video. When there exist unsatisfied subscribers, it means that the number of subscribers exceeds the system's capacity, and more server resources are required to accommodate all subscribers' requests. The heuristic can minimize the server bandwidth requirement, but the maximum number of subscribers that the system can accommodate is determined by the system itself. With this algorithm, we have the following desirable properties:

- Peers which help the system to provide bounded delay service are placed close to the server.
- Under the same conditions, subscriber peers have higher priorities than the ordinary peers to be pushed into the system. This is because the subscriber peers themselves are involved in calculating the fanout capacity.
- Ordinary peers also benefit from forwarding video to subscriber peers. They are placed close to the server, so they have a more reliable video stream with short delay.

The centralized heuristic design is used to analyze the optimal case, and is implemented for comparison.

6.3 Distributed Implementation

In practice, it is difficult to have complete knowledge about all peers. Further, the system cannot afford the cost of the aforementioned centralized method. We propose high-fanout promotion (HFP), a distributed delay-aware algorithm which helps the system to provide delay bounded videos to subscribers. In HFP algorithm, some peers get promoted to be closer to the server, so more subscribers get the video within the

delay bound T. The concept of fanout capacity can be found in Section III-B. The components of HFP algorithm are listed as follows.

6.3.1 New Peer Joining

The basic idea of our method is to utilize high fanout capacity peers to help delay-bounded video distribution. We make all peers maintain their bandwidth and playback delay information $< U_i, D_i >$ for the fanout capacity computation, and D_i is initially $+\infty$ when peer i joins the system.

Peer i contacts the server which maintains the list of online peers, and the server returns a few peers which have low playback delay as well as high residual bandwidth to it. If peer i is a subscriber peer, it checks the delay $d_{j,i}$ and the available bandwidth $r_{j,i}$ between itself and the potential parent peer j. It ignores those peers which have $D_j + d_{j,i} > T$.

Then it selects peers with low delay and high available bandwidth as its parents, e.g., peers with high $\frac{r_{j,i}}{D_j+d_{j,i}}$. Subscriber peer i might not get sufficient bandwidth (e.g., $\sum r_{j,i} < R$) because of the delay requirement. Then it contacts the server which compensates for the lack of bandwidth. The number of subscribers meets the upper bound of the system capacity when the server does not have bandwidth to serve subscribers. In this case, these subscribers and the ones after them will be treated as ordinary peers. If ordinary peers are not fully served by the peers returned by the server, they request the neighbors of those peers, and repeat the above process to find more parents.

6.3.2 High Fanout Peer Promotion

Peers with high fanout potential might join the system late, and their fanout capacities may be low because of being far away from the server and thus have long playback

delays. This is true for both subscriber peers and ordinary peers. In this section, we describe the algorithm to promote these peers so they can take over some subscriber peers' requests which are directly served by the server.

To get promoted, peers need to show that they would have higher fanout capacities if they were placed close to the server than the ones substituted by them. To figure out the fanout capacity, a peer maintains its playback delay, which can be done as following: whenever peer i receives a segment from the upstream peer j, it records the playback delay via peer j, namely, $D_j + d_{j,i}$. It periodically updates D_i using the Eq. (6.1). Please note that the delay recording and updating are independent. It is not necessary to update D_i whenever peer i receives a new segment. For example, we use the average recorded playback delay, and update D_i periodically. It reduces the computation cost and also avoids the deviation caused by traffic changes.

The promotion consists of two steps:

1) Detection: This step is used to find out if there are any upstream peers which can be swapped with a downstream peer. Normally, only peers with high upload capacities try to get promoted. Peer i contacts its upstream peer (e.g., peer j) within a fixed number of hops. Peer j returns its parent list to i, then i contacts them to find the link delays from these peers to itself and the playback delays at these peers. Then it figures out the playback delay D_i if peer i were placed at the position of peer j. It also figures out the new playback delay D_j at peer j when peer j is a subscriber. If the new playback delay still meets the bound, it is safe to place peer i at the position of subscriber j. Otherwise, the subscriber j's viewing experience will be affected.

Then peer i asks the server to send a list of subscriber peers which are physically close to peer i. Peer i contacts these peers and peer j's child subscriber peers to figure out how many subscriber peers it can serve if it were placed at the position of j, namely, the fanout capacity. It compares the fanout capacity with peer j's, and

peer j becomes a swapping candidate if its fanout capacity is smaller than peer i's.

After obtaining all the swap candidates, peer i chooses the one which has the lowest fanout capacity as the one to swap with. If no such candidate is found, peer i cannot be promoted.

2) Swapping: Peer i creates connections with parents of peer j, and informs peer j's subscriber peers to replace their parent j with it. At the same time, it disconnects from its current parents. Peer j also disconnects from its parents and connects peer i as its new parent peers. If it is not fully served by peer i, it connects to peer i's previous parents. To maintain smooth video playback during the swapping, the parent replacements are done one by one. For example, peer i connects to one of peer j's parents, but peer j is still connected to its parent until the connection between the parent and peer i is created and peer i is able to stream video content to the subscriber peers. Then peers j's previous subscriber peers disconnect from peer j.

6.3.3 Adaptation to dynamic cases

The promotion cost is non-trivial, and the promoted peers should serve for a long period. The promoted peer's stability affects the performance of our promotion algorithm. If a recently promoted peer leaves the system immediately, the subscriber peers cannot benefit from it, and their video quality is impaired. Further, it might need more server bandwidth to keep the video playback smooth.

Usually, the longer a peer resides in the system, the more stable it is [74]. In the implementation, we define the stability of a peer in terms of how long it has been, e.g., $log(t_i)$, and t_i is how long peer i is online. Peers which have high upload bandwidth, are physically close to some subscriber peers and have high stability values are the ideal candidates to submit the promotion requests.

6.4 Simulation Results

In this section, we evaluate the performance of our distributed HFP algorithm via simulations. We use the centralized heuristic as the benchmark. We also implement two other algorithms which are the greedy algorithm and the snow-ball algorithm [39]. The greedy algorithm is straightforward and directly places subscribers closer to the server compared to the ordinary peers. The snow-ball algorithm has close-to-optimum delay performance for the overall system, but it does not explicitly provide bounded delay to subscribers.

In the simulation, the link delays between peers are generated from the node to node latency measurement results in [68]. By default, the server's upload bandwidth is 3 Mbps and it streams out the video at 300 Kbps to 600 peers. The ratio among these settings is close to the real ratio presented in [79]. Peers dynamically join the system at an exponentially distributed rate with the average value of 10 peers/second. The peers have various upload bandwidth: 15% at 1 Mbps, 40% at 384 Kbps and 45% at 128 Kbps. 50% of all the peers are subscribers with 6-second delay bound, if not specified otherwise. We also run simulations with other parameters, and obtain results similar to those shown here. In the simulation, we check the subscribers' playback delay. The maximum number of subscribers that the system supports is equal to the number of subscribers with playback delay less than or equal to the delay bound. In other words, the system cannot serve all subscribers with delay-bounded video when the maximum playback delay at subscribers exceeds the bound.

We first investigate the performance of these algorithms with different session sizes and summarize the results in Figure 6.2. Usually, the overall playback delay in all algorithms increases with the increase in session size. The snow-ball algorithm does not differentiate between subscribers and ordinary peers, so there always exist bound violations at some subscribers. The results show that our algorithm can support 17%

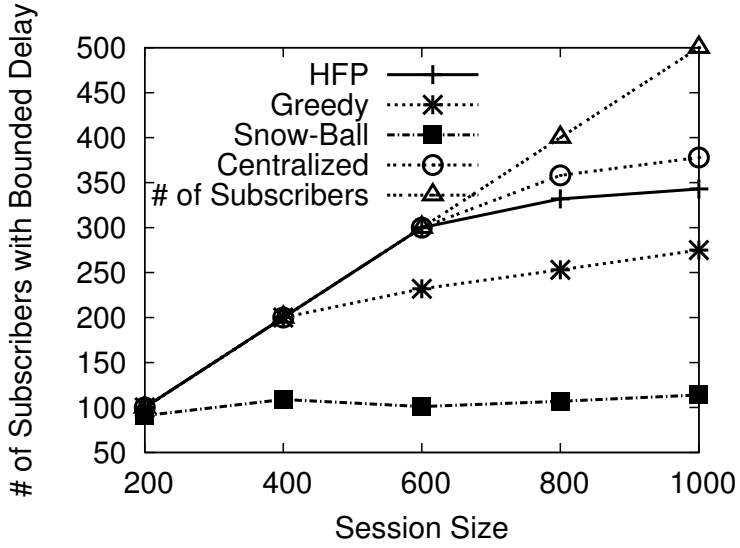

Figure 6.2: HFP enables the system of a larger size (600 peers) to provide all subscribers with delay-bounded (6 seconds) video, and it can support 17% to 50% more subscribers even if the system becomes too large to serve all subscribers with delay-bounded video.

to 50% more subscribers with bounded playback delay compared to the other two algorithms.

The snow-ball algorithm does not explicitly minimize the subscribers' delay, so it can only support a small-size session (e.g., < 200). When the session does not have many peers, the naive greedy algorithm is able to provide the video to subscribers within the 6-second bound. This is because the server's bandwidth is sufficient to compensate for the subscribers' requests. However, when the peers' population size increases, HFP algorithm outperforms the greedy algorithm as it organizes peers in a more efficient way.

Then we evaluate the impact of peers' bandwidth distribution. In Figure 6.3, we measure the maximum playback delay of all subscribers in two cases: peers with heterogeneous bandwidth and peers with homogeneous bandwidth. But we maintain the average bandwidth to be the same value in these two cases. We can see that the bandwidth heterogeneity helps all these algorithms achieve shorter subscriber play-

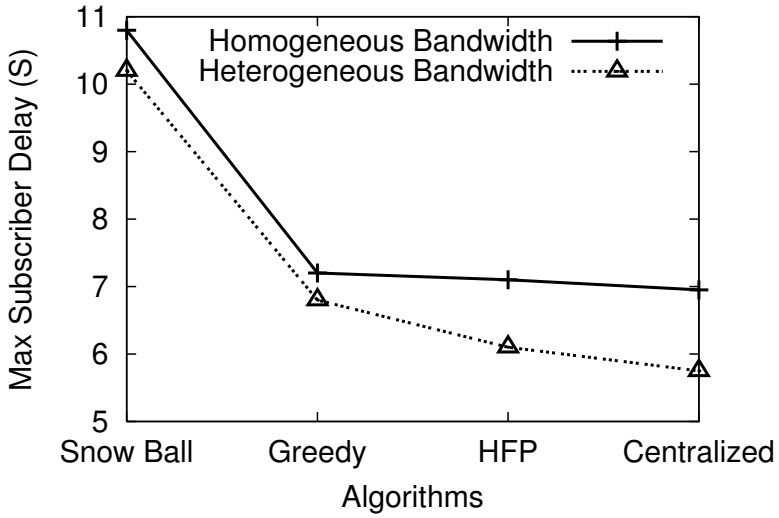

Figure 6.3: HFP works better in heterogenous bandwidth scenarios.

back delays. HFP algorithm can serve all subscribers with bounded-delay video (≤ 6 seconds) in the heterogeneous case. It uses ordinary peers to help reduce subscribers' playback delays, so the performance difference caused by bandwidth heterogeneity is more obvious in our algorithm than in the greedy algorithm and the snow-ball algorithm.

Figure 6.4 shows the subscriber percentage changes against playback delay. When the subscriber percentage is low, only a few peers have the delay-bounded requirement. Both the greedy algorithm and HFP algorithm can stream the video with bounded playback delay to all subscribers when the percentage of subscribers is low. When the subscriber percentage increases, both greedy algorithm and our algorithm cannot serve all subscribers within the bound, but our algorithm can serve about 17% more subscribers. As the snow-ball algorithm treats subscribers and ordinary peers the same, it is not affected by the percentage of subscribers. When all peers are subscribers, all of them have similar subscriber delays because now there is just a single type of peer in the system.

HFP algorithm dynamically promotes peers to new positions and connections are

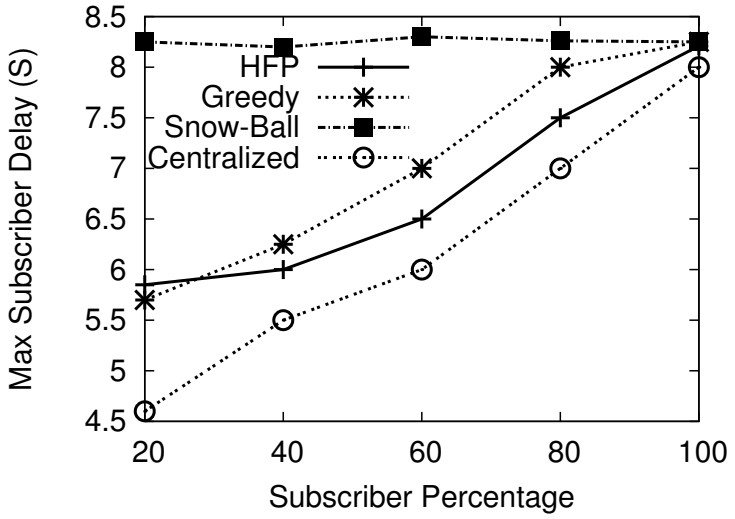

Figure 6.4: The more subscribers are, the more server bandwidth is required to provide the delay-bounded video.

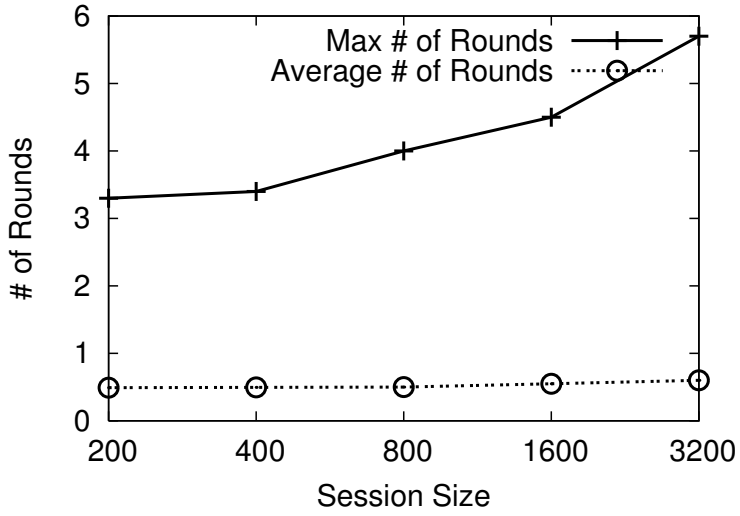

Figure 6.5: Convergence Speed of HFP: HFP converges fast even with the increase of system size.

re-established in the process. The convergence property is an important factor for evaluating our proposed HFP algorithm. We simulate the scenarios that all peers join the system within a short period and stay in the system until the end of the simulation. When a peer is promoted, we call it a promotion round. We evaluate the convergence through the promotion rounds. Figure 6.5 shows that the maximum number of promotion rounds per peer increases only a little when we double the session size, and the average value almost always remains low. This indicates that HFP algorithm converges fast to a stable state.

6.5 Conclusion and Discussion

Subscribers in a P2P live streaming system expect to view time-sensitive videos within a bounded delay. However, existing solutions do not provide the delay-sensitive peers called subscribers with bounded-delay service in a large-scale system. In this chapter, we study the problem of providing delay bounded videos to as many subscribers as possible. We show that the subscriber bounded delay (SBD) problem is NP-complete by relating it to the Steiner tree decision problem. Then we propose a heuristic and its decentralized implementation, called high fanout promotion (HFP) which helps the system with fixed server bandwidth to provide delay-bounded service to as many subscribers as possible. HFP algorithm promotes certain peer, which can serve many subscribers, close to the server, and therefore the system's capacity of serving subscribers is expanded. We conduct packet-level simulations to evaluate its performance, which show that HFP algorithm outperforms two other approaches. As future work, we will investigate a method to predict a system's capacity for serving subscribers and adopt admission control mechanisms in our solution.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this dissertation, we studied the design of efficient P2P streaming systems with heterogeneous user demands. Specifically, we considered the following three demand heterogeneity problems.

- To achieve user satisfaction and resource efficiency, P2P streaming systems should provide users with different streaming rates according to their demands on video quality (e.g., adaptive streaming rates). However, the demands are dynamic and a user may change his/her demand at anytime. The dynamic demand significantly complicates the design of a P2P streaming system. Thus we proposed algorithms to handle these heterogenous and dynamic demands.

- To provide time-shifted streaming service, we designed an efficient data prefetching algorithm and implemented it in a distributed way so that peers can share video content with one another.

- To provide a subset of peers called subscribers with a live broadcast program

within short and bounded playback delay is an NP Complete problem. We designed a heuristic algorithm to provide delay bounded videos to as many subscribers as possible.

When designing P2P streaming systems for demand heterogeneity, we should comprehensively consider several design goals (e.g., resource efficiency, user experience, system scalability, etc.). But these goals usually conflict with one another, and it is challenging to optimize all of them at the same time. A proper design should make reasonable trade-offs among these goals according to the specific application scenarios. Further, the implementation complexity is also an important factor. Our solutions use peer cooperation to achieve high streaming quality and low server cost, and the cooperation algorithms can be easily implemented. However, we do not have many real traces for demand heterogeneity simulation, since demand heterogeneity has received little attention and there are not many literatures about it. Therefore, we simulated as many typical demand heterogeneity scenarios as possible to evaluate our algorithms.

Peer cooperation is important in our algorithms, while the incentive mechanisms which encourage peers to contribute their bandwidth to help other peers are not discussed here. There are many literatures about incentive mechanisms, and they can be adopted into our algorithms with minor changes. This is beyond the scope of this dissertation. To sum up, our dissertation addresses how to efficiently utilize peers' resources to support heterogeneous demands.

7.2 Future Work

With the evolvement of hardware and wireless technology, mobile devices become powerful and thousands of millions users have been viewing multimedia content via

their mobile devices. Several commercial P2P streaming systems including UUSee [70], PPLive [58] and PPStream [60], also provide the mobile version of their P2P streaming software. We anticipate to see more and more mobile devices in P2P streaming systems because of their ubiquity. The future work is to extend the demand heterogeneity solutions to P2P streaming systems consisting of a large number of such mobile users. Recently, Liu *et al.* [37] investigate the optimal resource utilization in such P2P streaming systems by exploring all potentials of utilizing available peer resource. However, to fully accommodating demand heterogeneity in such systems is very challenging. The mobile devices have specific features such as mobility, limited power capacities, small screens, and so on. Users' subjective demands may not be fully supported because of the devices themselves. All these factors complicate the peer cooperation and the system design. To support users' heterogeneous demands in such systems requires significant modifications in current systems and further research.

P2P streaming has been proved successful in sustaining high definition video streaming with limited server bandwidth. The idea of peer cooperation can also be used in many other systems, such as content distribution network, massively multi-player online game, cloud computing. There are tremendous challenges and opportunities of extending P2P streaming technologies to these fields.

Bibliography

[1] `http://en.wikipedia.org/wiki/Zipf's_law`.

[2] M. Adler, R. Kumar, K. W. Ross, D. Rubenstein, T. Suel, and D. D. Yao. Optimal peer selection for P2P downloading and streaming. In *Proceedings of IEEE INFOCOM*, Miami, FL, March 2005.

[3] S. Agarwal and J. R. Lorch. Matchmaking for online games and other latency-sensitive p2p systems. In *Proceedings of SIGCOMM*, Barcelona, Spain, August 2009.

[4] P. Baccichet, J. Noh, E. Setton, and B. Girod. Content-aware P2P video streaming with low latency. In *IEEE Int. Conference on Multimedia and Expo, ICME*, Beijing, China, July 2007.

[5] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: High-bandwidth multicast in a cooperative environment. In *Proceedings of ACM SOSP*, Lake Bolton, NY, October 2003.

[6] J. Chakareski, S. Han, and B. Girod. Layered coding vs. multiple descriptions for video streaming over multiple paths. In *ACM Multimedia 2003*, Berkeley, California, November 2003.

[7] M. Chen, M. Ponec, S. Sengupta, J. Li, and P. A. Chou. Utility maximization in peer-to-peer systems. In *Proceedings of Sigmetrics*, Annapolis, Maryland, June 2008.

[8] X. Chen, M. Chen, B. Li, Y. Zhao, Y. Wu, and J. Li. Celerity: Towards low-delay multi-party conferencing over arbitrary network topologies. In *Proceedings of NOSSDAV*, Vancouver, Canada, June 2011.

[9] Y. R. Choe, D. L. Schuff, J. M. Dyaberi, and V. S. Pai. Improving VoD server efficiency with bittorrent. In *Proceedings of Multimedia*, Ausburg, Bavaria, Germany, September 2007.

[10] F. Clevenot-Perronnin and K. W. Ross. Multiclass P2P networks: Static resource allocation for service differentiation. *Performance Evaluation*, 62:32 – 49, 2005.

[11] Y. Cui and K. Nahrstedt. Layered peer-to-peer streaming. In *Proceedings of the 13th ACM NOSSDAV*, Monterey, CA, June 2003.

[12] Y. Cui, Y. Xue, and K. Nahrstedt. Optimal resource allocation in overlay multicast. *IEEE Transactions on Parallel and Distributed Systems*, 17(8):808–823, August 2006.

[13] S. Deshpande and J. Noh. P2TSS: Time-shifted and live streaming of video in peer-to-peer systems. In *Proceedings of IEEE Multimedia and Expo*, Hannover, June 2008.

[14] Amazon EC2. `http://aws.amazon.com/ec2/`.

[15] Susanna S. Epp. *Discrete Mathematics with Applications*. Brooks Cole Publishing Company, 2003.

[16] M. Ghanbari. Two-layer coding of video signals for VBR networks. *IEEE Journal on Selected Areas in Communications*, 7(5):771–781, June 1989.

[17] V. K. Goyal. Multiple description coding: Compression meets the network. *IEEE Signal Processing Magazine*, 18(5):74–93, September 2001.

[18] M. Guo and M. H. Ammar. Scalable live video streaming to cooperative clients using time shifting and video patching. In *Proceedings of IEEE INFOCOM*, Hong Kong, March 2004.

[19] Y. Guo, C. Liang, and Y. Liu. Adaptive queue-based chunk scheduling for p2p live streaming. In *Proceedings of IFIP Networking*, Singapore, May 2008.

[20] F. V. Hecht, T. Bocek, C. Morariu, D. Hausheer, and B. Stiller. Liveshift: Peer-to-peer live streaming with distributed time-shifting. In *Proceedings of IEEE P2P*, Maastricht, Netherlands, September 2008.

[21] C. Huang, J. Li, and K. Ross. Can Internet video-on-demand be profitable. In *Proceedings of ACM SIGCOMM*, Japan, August 2007.

[22] C. Huang, J. Li, and K. Ross. Can Internet video-on-demand be profitable. In *Proceedings of SIGCOMM*, Kyoto, Japan, August 2007.

[23] F. Huang, M. Khan, and B. Ravindran. On minimizing average end-to-end delay in P2P live streaming systems. *Lecture Notes in Computer Science*, 6490:459 – 474, 2010.

[24] Y. Huang, T. Fu, D. Chiu, J. Cui, and C. Huang. Challenges, design and analysis of a large-scale P2P-VoD system. In *Proceedings of ACM SIGCOMM*, Seattle, WA, August 2008.

[25] Y. Huang, T. Fu, D. Chiu, J. Cui, and C. Huang. Challenges, design and analysis of a large-scale P2P-VoD system. In *Proceedings of ACM SIGCOMM*, Seattle, WA, August 2008.

[26] Z. Huang, C. Mei, L. E. Li, and T. Woo. Cloudstream: delivering high-quality streaming videos through a cloud-based svc proxy. In *Proceedings of INFOCOM*, Shanghai, China, March 2011.

[27] F. Hwang, D. Richards, and P. Winter. The Steiner tree problem. North-Holland.

[28] H. Jagadish, B. Ooi, and Q. Vu. BATON: a balanced tree structure for peer-to-peer networks. In *Proceedings of the International Conference on Very Large Databases*, pages 661–672, 2005.

[29] B. Knutsson, H. Lu, W. Xu, and B. Hopkins. Peer-to-peer support for massively multiplayer games. In *Proceedings of IEEE INFOCOM*, Hong Kong, March 2004.

[30] B. Li, Y. Qu, Y. Keung, S. Xie, C. Lin, J. Liu, and X. Zhang. Inside the new Coolstreaming: principles, measurements and performance implications. In *Proceedings of IEEE INFOCOM*, Phoenix, AZ, April 2008.

[31] B. Li, Y. Qu, Y. Keung, S. Xie, C. Lin, J. Liu, and X. Zhang. Inside the new coolstreaming: Principles, measurements and performance implications. In *Proceedings of INFOCOM*, Phoenix, Arizona, April 2008.

[32] C. Liang, Z. Fu, Y. Liu, and C. W. Wu. iPASS: Incentivized peer-assisted system for asynchronous streaming. In *Proceedings of IEEE INFOCOM*, Rio de Janeiro, Brazil, April 2009.

[33] C. Liang, Y. Guo, and Y. Liu. Is random scheduling sufficient in P2P video streaming? In *Proceedings of ICDCS*, Beijing, China, June 2008.

[34] X. Liao, H. Jin, Y. Liu, M. Ni, and D. Deng. Anysee: Peer-to-peer live streaming. In *Proceedings of IEEE INFOCOM*, Barcelona Spain, April 2006.

[35] D. Liu, S. Chen, and B. Shen. PAT: Peer-assisted transcoding for overlay streaming to heterogeneous devices. In *Proceedings of NOSSDAV*, Urbana, Illinois, June 2007.

[36] D. Liu, S. Chen, and B. Shen. Dynamic bi-overlay rotation for streaming with heterogeneous devices. In *Proceedings of MMCN*, San Jose, California, January 2008.

[37] D. Liu, F. Li, and S. Chen. Towards optimal resource utilization in heterogeneous p2p streaming. In *Proceedings of ICDCS*, Montreal, Canada, June 2009.

[38] J. Liu, S. Rao, B. Li, and H. Zhang. Opportunities and challenges of peer-to-peer Internet video broadcast. *Proceedings of the IEEE*, 96(1):11–24, January 2008.

[39] Y. Liu. On the mimimum delay peer-to-peer video streaming: how realtime can it be? In *Proceedings of ACM Multimedia*, Augsburg, Germany, September 2007.

[40] Y. Liu, Y. Guo, and C. Liang. A survey on peer-to-peer video streaming systems. *Journal of Peer-to-Peer Networking and Applications*, 1(1):18–28, March 2008.

[41] Y. Liu, L. Xiao, X. Liu, L. Ni, and X. Zhang. Location awareness in unstructured peer-to-peer systems. *IEEE Transactions on Parallel and Distributed Systems*, 16(2):163–174, February 2005.

[42] Z. Liu, Y. Shen, S. S. Panwar, K. W. Ross, and Y. Wang. Using layered video to provide incentives in P2P live streaming. In *Proceedings of IPTV*, Kyoto, Japan, August 2007.

[43] Z. Liu, Y. Shen, S. Ross, S. Panwar, and Y. Wang. Substream trading: Towards an open P2P live streaming system. In *Proceedings of IEEE ICNP*, Orlando, FL, October 2008.

[44] N. Magharei, Y. Guo, and R. Rejaie. Issues in offering live P2P streaming service to residential users. In *Proceedings of IEEE Consumer Communications and Network Conference*, Las Vegas, January 2007.

[45] N. Magharei and R. Rejaie. PRIME: Peer-to-peer receiver-driven mesh-based streaming. In *Proceedings of IEEE INFOCOM*, Anchorage, Alaska, May 2007.

[46] N. Magharei, R. Rejaie, and Y. Guo. Mesh or multiple-tree: A comparative study of live P2P streaming approaches. In *Proceedings of IEEE INFOCOM*, Anchorage, AK, May 2007.

[47] L. Massoulie, A. Twigg, C. Gkantsidis, and P. Rodriguez. Randomized decentralized broadcasting algorithms. In *Proceedings of IEEE INFOCOM*, Anchorage, Alaska, May 2007.

[48] L. Massoulie, A. Twigg, C. Gkantsidis, and P. Rodriguez. Randomized decentralized broadcasting algorithms. In *Proceedings of IEEE INFOCOM*, Anchorage, AK, May 2007.

[49] J. Mol, D. Epema, and H. Sips. The Orchard algorithm: Building multicast trees for P2P video multicasting without free-riding. *IEEE Transactions on Multimedia*, 9(8), December 2007.

[50] NetFlix. `http://www.netflix.com`.

[51] J. Noh, A. Mavlankar, P. Baccichet, and B. Girod. Time-shifted streaming in a peer-to-peer video multicast system. In *Proceedings of IEEE Globecom*, Hawaii, November 2009.

[52] Z. Ouyang, L. Xu, and B. Ramamurthy. A partial forwarding scheme for dynamic window resizing in live P2P streaming systems. In *Proceedings of IEEE Globecom*, New Orleans, Louisiana, November 2008.

[53] Z. Ouyang, L. Xu, and B. Ramamurthy. Partial forwarding scheme for dynamic window resizing in live P2P streaming systems. In *Proceedings of IEEE GLOBECOM*, New Orleans, LA, November 2008.

[54] V. Padmanabhan, H. Wang, and P. Chou. Supporting heterogeneity and congestion control in peer-to-peer multicast streaming. In *Proceedings of IPTPS*, San Diego, CA, February 2004.

[55] V. Padmanabhan, H. Wang, P. Chou, and K. Sripanidkulchai. Distributing streaming media content using cooperative networking. In *Proceedings of NOSSDAV*, Miami Beach, Florida, May 2002.

[56] A. L. Peressini, F. E. Sullivan, and J.J. Jr. Uhl. The Mathematics of Nonlinear Programming. Springer, 1988.

[57] J. A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. H. J. Epema, M. Reinders, M. R. Van Steen, and H. J. Sips. Tribler: A social-based peer-to-peer system. In *Proceedings of IEEE IPTPS*, Santa Barbara, CA, February 2006.

[58] PPLive. `http://www.pplive.com`.

[59] PPLive. `http://www.pptv.com`.

[60] PPStream. http://www.ppstream.com.

[61] H. Jugen Proel and A. Steger. The Steiner tree problem : a tour through graphs, algorithms, and complexity. Friedrick Vieweg & Son, 2002.

[62] D. Ren, Y. H. Li, and G. Chan. Fast-mesh: A low-delay high-bandwidth mesh for peer-to-peer live streaming. *Multimedia, IEEE Transactions on*, 11:1446 – 1456, December 2009.

[63] D. Ren, Y. H. Li, and S. G. Chan. On reducing mesh delay for peer-to-peer live streaming. In *Proceedings of INFOCOM*, Phoenix, Arizona, April 2008.

[64] SopCast. http://www.sopcast.com.

[65] K. Sripanidkulchai, S. Sahu, Y. Ruan, A. Shaikh, and C. Dorai. Are clouds ready for large distributed applications? In *ACM SIGOPS Operating Systems Review archive Volume 44 Issue 2*, April 2010.

[66] Y. Sung, M. Bishop, and S. Rao. Enabling contribution awareness in an overlay broadcasting system. In *Proceedings of ACM SIGCOMM*, Pisa, Italy, September 2006.

[67] G. Tan and S. Jarvis. Inter-overlay cooperation in high-bandwidth overlay multicast. In *Proceedings of IEEE ICPP*, Columbus, OH, August 2006.

[68] Meridian: A Lightweight Approach to Network Positioning. http://www.cs.cornell.edu/People/egs/meridian/data.php.

[69] TVUPlayer. http://www.tvuplayer.com.

[70] UUSee. http://www.uusee.com.

[71] V. Venkataraman, K. Yoshida, and P. Francis. ChunkySpread: heterogeneous unstructured tree-based peer to peer multicast. In *Proceedings of IEEE ICNP*, Santa Barbara, CA, November 2006.

[72] V. Vishnumurthy and P. Francis. On heterogeneous overlay construction and random node selection in unstructured P2P networks. In *Proceedings of IEEE INFOCOM*, Barcelona, Spain, April 2006.

[73] F. Wang, J. Liu, and M. Chen. CALMS: Cloud-assisted live media streaming for globalized demands with time/region diversities. In *Proceedings of INFOCOM*, Orlando, Florida, March 2012.

[74] F. Wang, J. Liu, and Y. Xiong. Stable peers: Existence, importance, and application in peer-to-peer live video streaming. In *Proceedings of INFOCOM*, Phoenix, Arizona, April 2008.

[75] M. Wang and B. Li. Lava: A reality check of network coding in peer-to-peer live streaming. In *Proceedings of IEEE INFOCOM*, Anchorage, AK, May 2007.

[76] M. Wien, H. Schwarz, and T. Oelbaum. Performance analysis of SVC. *IEEE Transactions on Circuits and Systems for Video Technology*, 17(9), September 2007.

[77] C. Wu and B. Li. Diverse: application-layer service differentiation in peer-to-peer communications. *Selected Areas in Communications, IEEE Journal on*, 1(4):222–234, 2007.

[78] C. Wu, B. Li, and S. Zhao. Characterizing peer-to-peer streaming flows. *IEEE Journal on Selected Areas in Communications*, 25(9):1612–1626, December 2007.

[79] C. Wu, B. Li, and S. Zhao. Multi-channel live p2p streaming: Refocusing on servers. In *Proceedings of INFOCOM*, Phoenix, Arizona, April 2008.

[80] D. Wu, C. Liang, Y. Liu, and K. Ross. View-upload decoupling: A redesign of multi-channel P2P video systems. In *Proceedings of IEEE INFOCOM Mini-Conference*, Rio de Janeiro, Brazil, April 2009.

[81] D. Wu, Y. Liu, and K.W. Ross. Queuing network models for multi-channel live streaming systems. In *Proceedings of IEEE INFOCOM*, Brazil, April 2009.

[82] Y. Wu, C. Wu, B. Li, X. Qiu, and F. C. Liu. Cloudmedia: When cloud on demand meets video on demand. In *Proceedings of ICDCS*, Minneapolis, Minnesota, July 2011.

[83] X. Xiao, Y. Shi, Y. Gao, and Q. Zhang. LayerP2P: A new data scheduling approach for layered streaming in heterogeneous networks. In *Proceedings of IEEE INFOCOM*, Rio de Janeiro, Brazil, April 2009.

[84] Z. Xiao and F. Xie. New insights on Internet streaming and IPTV. In *Proceedings of the 2008 international conference on Content-based image and video retrieval*, Canada, July 2008.

[85] H. Xie, Y. Yang, A. Krishnamurthy, Y. Liu, and A. Silberschatz. P4p: Provider portal for applications. In *Proceedings of ACM SIGCOMM*, Seattle, WA, August 2008.

[86] Y. Liu and G. Simon. Peer-assisted time-shifted streaming systems: Design and promises. In *Proceedings of IEEE ICC*, Kyoto, Japan, June 2011.

[87] N. Yossef. A multiple-channel measurement study of PPStream - a commercial peer-to-peer media streaming system. *Master Project, Department of Computer Science and Engineering, University of Nebraska-Lincoln*, June 2008.

[88] Z. Ouyang and L. Xu and B. Ramamurthy. A cooperative scheme for dynamic window resizing in P2Plive streaming. In *Proceedings of IEEE ICC*, Dresden, Germany, June 2009.

[89] Z. Ouyang, L. Xu and B. Ramamurthy. A cooperative scheme for dynamic window resizing in P2P live streaming. In *Proceedings of IEEE ICC*, Dresden, Germany, June 2009.

[90] Z. Ouyang, L. Xu and B. Ramamurthy. On demand heterogeneity in P2P live streaming. In *Proceedings of IEEE ICC*, Cape Town, South Africa, May 2010.

[91] Z. Ouyang, L. Xu and B. Ramamurthy. Providing npr-style time-shifted streaming in P2P systems. In *Proceedings of IEEE ICCCN*, Maui, Hawaii, July 2011.

[92] Z. Ouyang, M. Wang, L. Xu and B. Ramamurthy. On providing bounded delay to subscribers in a P2P live streaming system. In *in submission*.

[93] M. Zhang, Y. Xiong, Q. Zhang, and S Yang. On the optimal scheduling for media streaming in data-driven overlay networks. In *Proceedings of IEEE Globecom*, San Francisco, California, November 2006.

[94] M. Zhang, Y. Xiong, Q. Zhang, and S. Yang. Optimizing the throughput of data-driven peer-to-peer streaming. *IEEE Transactions on Parallel and Distributed System*, 20(1):97–110, 2009.

[95] M. Zhang, Y. Xiong, Q. Zhang, and S. Yang. Optimizing the throughput of data-driven peer-to-peer streaming. *IEEE Transactions on Parallel and Distributed Systems*, 20(1):97–110, Januray 2009.

[96] M. Zhang, Q. Zhang, and S. Yang. Understanding the power of pull-based streaming protocol: Can we do better? *IEEE Journal on Selected Areas in Communications*, 25(8):1678–1694, 2007.

[97] X. Zhang, J. Liu, B. Li, and T. Yum. DONet/CoolStreaming: A data-driven overlay network for live media streaming. In *Proceedings of IEEE INFOCOM*, Miami, Florida, March 2005.

[98] R. Zimmermann and L. S. Liu. Active: Adaptive low-latency peer-to-peer streaming. In *In Proc. of SPIE/ACM Multimedia Computing and Networking*, 2005.