On Reducing Delays in P2P Live Streaming Systems

Fei Huang

PhD Dissertation Proposal submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Electrical and Computer Engineering

Binoy Ravindran, Chair
Yiwei Thomas Hou
E. Douglas Jensen
Subhash C Sarin
Yaling Yang

October 16, 2009
Blacksburg, Virginia

Keywords: P2P Streaming, Quality of Service (QoS), Optimization, Reliability

On Reducing Delays in P2P Live Streaming Systems

Fei Huang

(ABSTRACT)

Peer-to-peer (P2P) technology provides a scalable solution in multimedia streaming. Many streaming applications, such as IPTV and video conferencing, have rigorous constraints on end-to-end delays. Obtaining assurances on meeting those delay constraints in dynamic and heterogenous network environments is a challenge. In this proposal, we devise a streaming scheme which minimizes the maximum end-to-end streaming delay for a mesh-based overlay network paradigm. We first formulate the minimum-delay P2P streaming problem, called the MDPS problem, and prove its NP-completeness. We then present a polynomial-time approximation algorithm to this problem, and show that the performance of our algorithm is bounded by a ratio of $O(\sqrt{\log n})$. For a practical deployment, we extend the algorithm to a distributed version with the adaptation to network dynamics. Our simulation study reveals the effectiveness of our algorithm, and shows a reasonable message overhead.

Besides playback lag, delays occurring in P2P streaming may arise from two other factors: node churn and channel switching. Considering the fact they both stem from the re-connecting request in churn, we call them churn-induced delay. A typical channel switching delay costs around 10 seconds or even more. Current users have been accustomed to delays under seconds, which are typical in a cable TV system. Thus, this long switching delay has negatively affected the extensive commercial deployment of P2P systems. In this proposal, we propose an agent-based scheme to provide preventive connections to all channels. Once an actual connection is requested, time will be saved in retrieving bootstrapping information and obtaining authorization as well as authentication in new channel. Towards an efficient control and message overhead, each channel will select some powerful peers as agents to represent the peers in the channel. Agents will distill the bootstrapping peers with superior bandwidth and lifetime expectation to quickly serve the viewer in the initial period of streaming. We build a queueing theory model to analyze the agent-based scheme. Based on this model, we numerically compare the performance of our scheme with previous general scheme. The results of numerical experiments justify the significant performance of our scheme.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problems and Motivation

In the recent decade, P2P networks have greatly enhanced content distribution on the Internet by enabling efficient cooperation among end users [1, 2]. Benefiting from its significant scalability, there is an increasing demand on applications of P2P live streaming, such as IPTV, VOIP, and video conferencing [3–6]. As a result, many P2P live video systems, such as PPLive, Coolstreaming, and Sopcast, etc., have been successfully deployed in the recent years, and most of them have over 100 channels, prevailing with millions of users [7, 8]. However, recent measurement studies of P2P streaming indicate the detrimental impacts from node churn, long switching delay, and playback lag, have hindered the extensive commercial deployment of P2P systems [9, 10]. Motivated by this, in this proposal, we work towards reducing several types of streaming delay in P2P networks.

First, we will focus on minimizing playback lag, i.e. end-to-end streaming delay. As we know, in many classes of P2P streaming applications, the delivery of real-time video content imposes rigorous constraints on the end-to-end delay. Obtaining assurances on meeting such delay constraints in highly dynamic and heterogenous P2P network environments is a

challenging and open problem. The streaming delay has negatively affected the extensive commercial deployment of P2P systems. For example, IPTV deployment from commercial service providers is far below the industry expectation [1]. Motivated by this, our first target is to minimize the end-to-end streaming delay in P2P networks.

Minimizing streaming delays in P2P networks is an NP-complete problem. This is due to heterogeneous bandwidth requirements and network dynamics of P2P systems. Thus, obtaining optimal solutions to this problem for large-scale networks is intractable. In this proposal, we propose an efficient approximation algorithm for this problem, which *provably* achieves a delay assurance by an approximation ratio of $O(\sqrt{\log n})$ where n represents the number of peers in the network. Based on an analytical model, we then design an adaptive distributed version of the algorithm, which can be easily deployed in a fully dynamic network environment.

Previous work on P2P streaming can be broadly classified into two classes: (1) multiple tree-based overlays, and (2) mesh-based overlays [8, 11–15]. Recent studies have shown that the mesh-based approach consistently exhibits a superior performance over the tree-based approach [12, 16]. The tree-based P2P streaming approach organizes peers into multiple diverse trees. After obtaining the description of a Multiple Description Coded (MDC) content, it pushes each description through separate trees [11, 12]. In contrast, the mesh-based P2P streaming approach arranges peers into a randomly connected mesh and employs swarming content delivery [12]. The major advantages of mesh-based systems are easy maintenance and inherent robustness in high-churn P2P environments [8]. Motivated by these promising advantages, we study the minimum-delay problem under the mesh-based model. Although our algorithm is developed with a mesh paradigm, our study also indicates its readiness to fit the multiple tree-based model after simple modifications. To reduce the complexity of the problem, our proposal focuses only on minimizing the communication delay. For packet scheduling, there exists a vast array of solutions, such as [15, 17, 18]. The mesh built from our algorithm can adopt any of these scheduling algorithms to yield low-delay streaming.

Existing heuristics on the problem of reducing P2P streaming delay either provides no theoretical bound on the worst-case performance or loosely estimate the bound without a robust theoretical analysis [13, 14, 19]. The estimated bound of previous algorithms [19, 20] is $O(\log n)$. In this proposal, we not only present an approximation algorithm with a strong theoretical basis, but also reduce the approximation factor to a ratio of $O(\sqrt{\log n})$.

Second, besides playback lag, delays occurring in P2P streaming may arise from two other factors: node churn and channel switching. Node churn represents the frequent event of peer arrival or departure. Typically, peer departure can lead to streaming interruption to the downstream peers, where delay will happen in downstream peers during the restoration of streaming connections [9]. We call such delay as *recovery delay*. Another type of churn is channel churn, which represents the event of channel switching in multi-channel P2P streaming system. It has been found that channel churn is much more frequent than node churn [7], which brings severe instability to the system. In multi-channel P2P streaming system, the video content of each channel is distributed by a different swarm of peers. Similar to the node churn, when a peer switches from channel A to channel B, its downstream peers need to locate new data feed from other peers in the swarm of A. Additionally, channel switching delay occurs as well when this peer attempts new stream connections in swarms of B. As we can see, all those types of delay stem from the churns of node departure and channel switching. We generally call such delay *churn-induced delay*.

Current users have been accustomed to delays under seconds, which are typical in a cable TV system [7]. However, churn-induced delays are significant in current P2P system. For example, measurement studies indicate that channel switching delays are typically on the scale of 10-60 seconds [8]. From the perspective of user experience, this is obviously undesirable. Motivated by this, we propose a novel agent-based P2P architecture to preventively reduce the churn-induced delay. The scheme built from this proposal can adopt our minimum-playback-delay scheme in this proposal [21] or any of the existing algorithms, such as [14, 22], to yield low-delay streaming.

In this proposal, we consider the fact that churn-induced delays identically stem from the time of re-connecting to new peers. Thus, our scheme propose preventive connections to all channels. Once an actual connection is requested, time will be saved in retrieving bootstrapping information and obtaining authorization as well as authentication. However, maintaining preventive connections for all peers to all channels makes it impractical in terms of the enormous control signals and message overhead. Towards an efficient control and message overhead, we propose that each channel will select some powerful peers as agents to represent the peers in the channel. Agents will distill the bootstrapping peers with superior bandwidth and lifetime expectation to quickly serve the viewer in the initial period of streaming. Moreover, agent will pre-schedule the downloading plan about data chunk partition and streaming for future viewers, and coordinate with each other to build the preventive connections for peers represented by them.

1.2 Summary of Current Research and Contributions

In summary, our main goal in current research is to design the schemes for reducing the major delays occurring in P2P streaming. Our work makes the following important contributions:

(1) To the best of our knowledge, our work represents the first approximation algorithm that optimizes P2P streaming delay. The result of this scheme provides an provable upper bound of $O(\sqrt{\log n})$. First, we partition the peers V into different clusters according to the regional aggregated streaming capacities. Then, we filter the peers by keeping one representative peer for each cluster, whose streaming capacity is virtualized by the aggregated streaming capacities of the entire cluster. As a result, we form a backbone using the representative nodes. Since the backbone nodes are virtual representatives of clusters, in the next step, we expand the mesh connections from the representative nodes into clusters, which constitutes a final streaming mesh for the overlay network.

(2) We analyze the approximation algorithm's performance and derive the algorithm's

approximation ratio, which is found to be the lowest ratio when compared with past results. We not only present an approximation algorithm with a strong theoretical basis, but also reduce the approximation factor to a ratio of $O(\sqrt{\log n})$.

(3) We extend the approximation algorithm to a practical distributed version that is robust to high user churn. Our simulation results indicate our algorithm can actively ensure the end-to-end streaming delay in the worst-case scenario.

(4) Our work on reducing the churn-induced delay provides the first scheme in this topic with analytical model and numerical evaluation on the performance. Our scheme proposes preventive connections to all channels. Once an actual connection is requested, time will be saved in retrieving bootstrapping information and obtaining authorization as well as authentication. Our scheme suggests an idea of agent, which facilitates the bootstrapping process in channel switching and peer recovery.

(5) We build a queueing theory model for the agent-based scheme. Based on this model, we theoretically analyze the performance of our scheme.

(6) We analyze the scheme's performance and derive the scheme's optimal settings based on the numerical experiments. Our numerical experiment results indicate our scheme can significantly reduce the churn-induced delay, especially for the channel-switching delay.

1.3 Summary of Proposed Post-Preliminary-Exam Work

Based on these research results, we propose the following major research directions for the post-preliminary-exam work:

(1) *Minimize the end-to-end streaming delay in multi-channel scenario.* We propose to devise an algorithm to minimize the end-to-end streaming delay specifically working on multi-channel scenario. Our current work on the problem of reducing P2P streaming delay only focus on the single-channel scenario. However, in most P2P streaming applications,

there commonly exists hundreds of channels [7]. As we know, bandwidth allocation and peer assignment in multi-channel streaming pose extra challenge to the minimum-delay problem. Furthermore, current algorithms on the single-channel scenario, if applied in the multi-channel scenario, either provides no theoretical bound on the worst-case performance or loosely estimate the bound without a robust theoretical analysis [13, 14, 19]. The estimated bound of those previous algorithms [19, 20] is $O(\log n)$. However, $O(\log n)$ is trivial as any depth first search tree will provide such delay bound. Several directions can be considered to reduce the multi-channel streaming delay. For example, by smartly utilizing the bandwidth allocation between channels, we may balance the resource among channels so as to improve the worst delay in less popular channel and provide a minimum average delay on viewers of all channels. We propose to design improved approximation-algorithm-based protocols which considers the optimal bandwidth allocation to achieve a minimum streaming delay. An example design is to decouple the viewing and uploading functionalities on the viewer and reassign the uploading bandwidth in popular channel to improve the streaming delay in less popular channel. By leveraging our current clustering-based approximation, we may guarantee an approximation bound when allocate the bandwidth optimally among channels.

(2) *Design a delay-optimized admission control algorithms.* Industrial deployment of P2P streaming applications may involve different types of peers, including free user, ordinary member paying fees to watch specific channel and VIP member paying extra fees for high-quality service. However,the traditional scheme in cable TV to prioritize VIP member when bandwidth resource is not sufficient is not applicable. For example, some free users may have excellent bandwidth to contribute. If we simply put them at the edge of streaming mesh with worst service, we may lost the opportunity to rationally utilize such bandwidth. We propose to devise an admission control algorithm specifically working on multi-type peers. An example solution is to put different reward values when satisfying the service of different types of users and also put rewards based on the streaming delays and bandwidth resources they have. An optimal solution on reward may well utilize the peers with high bandwidth even they are free or low-level users.

In addition to these major directions, we also propose the following minor research directions:

(3) *Design a large-scale simulation based on our algorithms.* P2P networks typically involves hundreds of thousands of users. To evaluate our schemes on large-scale applications, traditional simulation tool like ns-2 is not applicable with memory issues. Some possible solutions emerges as simulation tool exclusively for P2P simulation. For example, PeerSim may handle a simulation for over half-million peers. Such large-scale simulation can evaluates the performance of our distributed scheme with real scalability and may help us identify some improvement directions on the schemes.

(4) *Minimize the scheduling delay algorithms.* P2P streaming need to exchange data chunk and decide which chunk should transfer with priority. Delayed receiving of chunk may incur delays or interruptions in streaming. So we propose to design a chunk scheduling algorithm to maximize the probability that a chunk will arrive to the viewer before it need to be played. One example solution is that seldom existed chunk should be transferred with priority so more peers may span out this chunk to start a smooth streaming.

(5) *Design a gossip-based message exchange scheme.* In our current study, gossip is proposed to exchanging agent data and search for resources. We propose to design an improved gossip scheme to ensure a time constraints on message receipts. One possible solution is to utilize super-peers to manage the normal peers and exchange resource information with them. So gossip only needs to run through these super-peers. Since super-peers have long expected lifetime, i.e. less probability to departure, we can ensure the time constraints in resource searching with certain probability guarantee.

1.4 Outline

The rest of this proposal is organized as follows. Chapter 2 describes an overview of past and related works. In Chapter 3, we describe and formulate the minimum-delay P2P streaming

problem. We prove its NP-completeness and then presents our proposed approximation algorithm and derive its performance guarantee. We compare our algorithm against past works through simulation-based experimental studies. Chapter 4 describe the problem of reducing churn-induced delay, i.e. channel-switching delay and recovery delay, and propose our agent-based scheme. We model our scheme by queuing theory model. Based on that, we numerically study the performance our scheme against general scheme. Chapter 5 summarizes the work in this proposal and describes our post-preliminary-exam work.

Chapter 2

Related Work

First, we describe the related work for the minimum end-to-end streaming delay problem. Most of the previous protocols for the P2P streaming delay problem are based on tree-shaped overlays [6, 11, 13, 23, 24]. Tran *et al.* propose Zigzag in [23], an approach to cluster peers into a hierarchy, called the administrative organization, for easy management, and build the multicast tree atop this hierarchy so as to reduce the delay. In [13], Noh *et al.* propose an overlay consisting of multiple trees with moderate out-degree to reduce end-to-end transmission delays in P2P media streaming systems. In [11], Venkataraman *et al.* present Chunkyspread, which splits a stream into distinct slices and transmits them over multiple trees by a P2P multicast algorithm. However, such multiple-tree overlay construction cannot be easily maintained [12]. In addition, it is difficult to obtain the globally optimal delay and coordinate among different trees [14]. Moreover, resource utilization of multiple-tree approaches is generally speaking, relatively low. For example, all leaf nodes do not contribute any bandwidth or CPU cycles to the multicast trees [25].

Recently, an increasing number of studies have focused on mesh-based P2P live streaming [14,15,20,22]. In [14], Ren *et al.* propose a heuristic to reduce the delay on mesh topology, where peers select their parents based on the metric of link capacity divided by the communication delay. In this algorithm, peers located at the edge of mesh may only download

the data without contributing its bandwidth resource, which may lead to low bandwidth utilization in the network. Thus, when the total uploading capacity is close to the downloading capacity among peer nodes, some peers may not be able to receive a live streaming. Besides, the heuristic does not provide performance guarantees on the end-to-end streaming delay, which is critical in delay-sensitive applications, such as video conferencing. In [22], Wu *et al.* present a distributed algorithm for obtaining the optimal average streaming delay. They apply several techniques in linear programming, such as Lagrangian relaxation and the subgradient algorithm. To reduce the computational complexity, they strictly limit the potential connections for each peer, which may restrict its practical applications. In their experimental results, we can observe significant time costs for achieving a near-optimal result. For a large-scale network, the convergence of the algorithm cannot be guaranteed, which may significantly increase the P2P start-up delay. Moreover, to exchange computational data between peers, considerable message overhead may be incurred in the network.

Minimum-delay P2P streaming has some similarity with the minimum-delay multicast tree problem (MDMT problem) [26, 27] and degree-bounded minimum-diameter tree problem (DBMDT problem) [28–30]. Our algorithm is inspired by the clustering method first proposed by Könemann *et al.* [28]. However, previous approximation algorithms on MDMT and DBMDT generally assume a constant and equivalent degree on all the nodes and consider a single-commodity flow for each receiver. These assumptions are not appropriate for P2P streaming, where peers have heterogeneous and dynamic bandwidth capacities, i.e., heterogeneous degrees on the nodes, and they need to aggregate the multiple flows for a smooth playback. Toward that end, the clustering in our algorithm will generate a subgraph which is not simply a tree. Besides, the proofs on DBMDT theorems in [28] highly depend on the equality of node degree, which will cause the major proofs in their work do not hold in our case. Moreover, the DBMDT problem in [28] is bounded by a bidirectional degree, which does not distinguish out-degree and in-degree. For example, when calculating the aggregated cluster degree, their method, if used in our paradigm, will depend on both in and out degrees; however, only the out-degree should be counted in this scenario. All those

differences make our problem more challenging than MDMT and DBMDT. In particular, DBMDT is a special case of the MDPS problem, in which every node has uniform outlink capacity and an inlink capacity of 1.

On the following, we introduce the related work on reducing churn-induced delay. Previous works in IPTV indicate the next channel a user will watch can be predicted with the probability analysis based on user switching pattern [31–34]. For example, a viewer watching a live news channel may switch to another news channel with high probability. Also, it is highly probable that a viewer will switch to the adjacent channel in the channel list when he is seeking for some interesting channel in sequence. Thus, several papers, such as [31, 32], in multicast IPTV system propose to send contents of the predicated next channels in parallel with the currently viewed channel. Should the user switch to one of the predicated channels, he can immediately watch the next channel without delay. However, such method is bandwidth consuming in transmitting multiple streams, which is not practical for the current P2P streaming system due to the limited upload and download bandwidth resource.

Channel-switching delay problem has high similarity with segment-seeking delay problem in video-on-demand (VoD), where users feel delay when they're browsing the channel for a segment they are interested in. Segment-seeking delay is defined as the interval between the time requesting a segment and the time when the segment is ready for playback [35]. Existing studies [35, 36] in video-on-demand (VoD) reveal prefetching algorithms with segment prediction can be effective to reduce seeking delay. Prefetching scheme streams one or multiple predicted segments while the current segment is being played [35]. For example, in [35], He *et al.* presents an optimal prefetching scheme and an optimal cache replacement policy to minimize the expected seeking delay at every viewing position. Their scheme captures the seeking statistics, and based on that, estimates the segment access probability. Clearly, this method can be extended to the multi-channel VoD system by prefetching predicted channels [8]. However, such methods can not be extended to live channel applications, where prefetching of live video content is not applicable.

The algorithm of reducing churn-induced delay in P2P streaming has close relation to the problem of improving the churn resilience because a resilient system generally has less streaming interruptions [7, 37–40]. In [7], Wu propose a strategy, called view-upload decoupling (VUD), to reduce the channel churn. The VUD-based P2P streaming approach decouples the role of peer into viewing and uploading. In another word, what a peer uploads is independent of what it views [9]. The major advantages of VUD-based systems are inherent robustness in the high-channel-churn environments and flexible cross-channel resource sharing capability. The drawback of this approach comes from the fact VUD costs more bandwidth overhead. To tackle this problem, Wu *et al.* propose a substream-swarming enhancement [9]. Specifically, each swarm only distributes a small portion of the stream, called substream, to viewers. Viewers needs to collect substreams from multiple swarms for a complete streaming. This substream concept can improve the viewing performance without significant bandwidth overhead. As we can see, in VUD, the distribution swarms are more stable than traditional system where peers deliver the same channel content they are viewing. As a churn-resilient scheme, the channel-switching behavior of a peer in VUD will not influence its downstream peers. Yet, on the other side, VUD can neither reduce the delay consumed at the peer that is switching the channel, nor the delay from the node churn.

In stead of mitigating one type of delay, our strategy focuses on retrenching the connection time for all cases of churn-induced delay. Additionally, considering the promising advantages in VUD, we can integrate our scheme with VUD to obtain better performance.

Chapter 3

Minimum-Delay P2P Streaming

In this chapter, we describe the problem of minimum-delay P2P streaming, i.e. minimum playback lag problem. First, we formulate the minimum-delay P2P streaming problem and prove its NP-hardness. Then we present our proposed approximation algorithm and justify its sub-optimum performance assurance. Simulation results are demonstrated at the end of this chapter.

3.1 Problem Formulation

In this section, we formally state the minimum-delay P2P streaming problem (MDPS problem) and show that the problem is NP-complete.

3.1.1 Minimum-Delay P2P Streaming Problem

We model an overlay network as a directed graph $G = (V, E)$, where V is the set of vertices representing peer nodes, and E is the set of overlay edges representing directed overlay links. Let n represent the number of peers in the network, i.e. $n = |V|$. Each overlay link $(i, j) \in E$ is associated with a communication delay $d_{(i,j)}$. In the rest of this proposal, we

define the length of edge (i, j) as $d_{(i,j)}$, $\forall(i, j) \in E$. We assume that G is symmetric, i.e., $d_{(i,j)} = d_{(j,i)}, \forall i, j \in V$, and the delays associated with G form a metric, i.e., G satisfies the triangle inequality. For every peer $i \in V$, we define an upload capacity of O_i units/second and a download capacity of I_i units/second. For ease of presentation, we define *unit* as the minimum package size in P2P streaming, which varies in different applications [5, 41].

We consider a peer-to-peer streaming session to originate from a single source node S to a set of receivers R, where $V = \{S\} \cup R$. Peers may receive the streaming data from the source node directly or indirectly from multiple P2P paths. Suppose S streams data at a constant streaming rate of s units/second. We denote f_{ij} as the rate at which peer i streams to peer j. If peer j receives the aggregated stream at s units/second from its parents, we call peer j as *fully served* [14]. Mathematically, the fully served requirement of peer j can be expressed as $\sum_{i:i\in L_j} f_{ij} = s$, where L_j is the set of parents of peer j. We assume that a fully served peer can smoothly play back the streaming content at its original rate of s units/second [14].

We call the stream from the source to one receiver j as the *P2P unicast flow* to j. A P2P unicast flow U may consist of streams from multiple P2P paths, called *fractional flows* [22]. Each fractional flow $p \in U$ has the arrival latency t_p from the source to receiver, where $t_p = \sum_{(i,j)\in p} d_{(i,j)}$. The latency of the unicast flow U can be defined as the maximum latency among its fractional flows, i.e., $\max_{p\in U} t_p$. To stream multimedia content to multiple receivers, we can envision multiple unicast flows from the source to receivers. Thus, the maximum delay in P2P streaming is defined as the maximum latency of all unicast flows.

We now define the problem formally:

Definition 1. ***Minimum-Delay P2P Streaming Problem (MDPS problem):*** *Given the constraints that are previously mentioned, the MDPS problem is to devise a streaming scheme which minimizes the maximum end-to-end streaming delay with each receiver fully served.*

To help obtain greater insights about the MDPS problem, we formulate the problem in the integer linear programming framework, as follows:

$$\min\ t \tag{3.1}$$

subject to

$$\sum_{(i,j)} d_{(i,j)} x^r_{ijm} \leq t, \forall (i,j) \in E, \forall r \in R, \forall m \tag{3.2}$$

$$x^r_{ijm} \in \{0,1\}, \forall (i,j) \in E, \forall r \in R, \forall m \tag{3.3}$$

$$x^r_{ijm} \geq f^r_{ijm}/s, \forall (i,j) \in E, \forall r \in R, \forall m \tag{3.4}$$

$$s \geq f^r_{ijm} \geq 0, \forall (i,j) \in E, \forall r \in R, \forall m \tag{3.5}$$

$$f^r_{ijm} - f^r_{jim} = b^r_{im}, \forall (i,j) \in E, \forall r \in R, \forall m \tag{3.6}$$

$$\sum_{m=1}^{s} \sum_{r:r\in R} f^r_{ijm} \leq y_{ij}, \forall (i,j) \in E \tag{3.7}$$

$$\sum_{j:(i,j)\in E} y_{ij} \leq O_i, \forall i \in V \tag{3.8}$$

$$\sum_{j:(j,i)\in E} y_{ji} \leq I_i, \forall i \in V \tag{3.9}$$

$$0 < m \leq s, \tag{3.10}$$

where

$$b^r_{im} \begin{cases} \geq 0 & \text{if } i = S, \\ \leq 0 & \text{if } i = r, \\ = 0 & \text{otherwise,} \end{cases} \tag{3.11}$$

$$\sum_m b^r_{im} = \begin{cases} s & \text{if } i = S, \\ -s & \text{if } i = r, \\ 0 & \text{otherwise.} \end{cases} \tag{3.12}$$

In this integer programming (IP) expression, t denotes the streaming delay of the fractional flow; x_{ijm}^r is set to 1 only if there is a connection between peer i and j on the m^{th} fractional flow to receiver r; f_{ijm}^r represents the m^{th} fractional flow rate on link (i, j) to receiver r; and y_{ij} is the aggregated flow rate on link (i, j).

The exact solution to this problem is optimal, but is computationally intractable to determine, and not practical in real applications. We will now show the NP-completeness of this problem, which motivates us to develop a near-optimal approximation algorithm.

Lemma 1. *If the instance of MDPS problem has a solution, then the sum of the upload capacities, including source and receivers, must be no less than the sum of fully served streaming rates at all receivers, i.e.,*

$$\sum_{i \in V} O_i \geq (|V| - 1) \times s. \tag{3.13}$$

Proof. Suppose we have a feasible streaming scheme described by the graph $A = (V, E_f)$, where $E_f \subset E$ represents the P2P connections among V. We can envision that each peer $v \in V$ consists of two conceptual nodes v_{in} and v_{out}, where v_{in} represents the download behavior, and v_{out} represents the upload behavior. Thus, A can be envisioned as a bipartite graph $A' = (V_{in}, V_{out}, E_f)$, where V_{in} is the set of all v_{in} and V_{out} is the set of all v_{out}. Now, the flow out of V_{out} should be equal to the flow into V_{in}, i.e., $(|V| - 1) \times s$. Since the flow out of V_{out} cannot exceed its total upload capacities, we have $\sum_{i \in V} O_i \geq (|V| - 1) \times s$. The lemma follows. □

According to Lemma 1, it is reasonable to assume that the preliminary condition in Equation (3.13) holds. In addition, we presume that the download capacity $I_i \geq s, \forall i \in V$ for a smooth playback at receivers.

3.1.2 Hardness

Theorem 1. *The minimum-delay P2P streaming problem is NP-complete.*

Proof. We first show that the MDPS problem can be reduced from the Freeze-Tag Problem (FTP), which is well known to be an NP-hard problem [42, 43]. Given a complete graph $G' = (V', E')$ in metric space, FTP can be described by a set of robots V', among which one robot S' is initially awake and all the others R' are asleep. The robot which is awake will select B' sleeping robots and awaken them. Once awaken, each new robot is available to assist in rousing another set of B' robots. There exists a latency between each pair of robots. Thus, a solution to the FTP can be described by a wake-up tree T which is a directed B'-ary tree rooted at S', spanning all robots R'. The objective is to minimize the makespan of T, i.e., the time t' when the last robot awakens [42].

Let the formulation of G in MDPS be identical to G' in FTP, and let the streaming rate be $s = 1$ units/second. For all nodes V in G, we set their upload capacities to be B' units/second. In this case, every node will have exactly one parent and can stream up to B' children. As we can observe, the resulting topology becomes a B'-ary spanning tree T, and the maximum end-to-end streaming delay of t is the latency of the last peer in T. Therefore, the optimal solution to the MDPS problem can also solve the FTP problem optimally. In other words, the FTP will have a tree T in G' with minimum delay of $\min t'$ if and only if the same T in G provides the minimum-delay spanning tree and the resulting min-max delay of MDPS equals $\min t'$. Since FTP can be reduced to the MDPS problem, the MDPS problem is NP-hard. Also, we can clearly see MDPS is in NP since it is easy to check whether a streaming scheme has a delay of t and follows the streaming constraints listed in Equations (3.8), (3.9), (3.12). Thus, we complete the proof. □

3.2 Approximation Algorithm

3.2.1 Overview of Techniques

Given the NP-complete nature of the MDPS problem, our goal is to design an efficient polynomial-time approximate solution with a provable performance bound. Our work builds on by Könemann *et al.* [28] and we extend a number of their concepts, including *clustering* and *filtering*. First, we partition the peers V into different clusters according to the regional aggregated streaming capacities. Then, we filter the peers by keeping one representative peer for each cluster, whose streaming capacity is virtualized by the aggregated streaming capacities of the entire cluster. As a result, we form a backbone using the representative nodes. Since the backbone nodes are virtual representatives of clusters, in the next step, we expand the mesh connections from the representative nodes into clusters, which constitutes a final streaming mesh for the overlay network.

To simplify the complexity of the problem, we assume that at least half of the nodes have upload capacities $O_i \geq 2s$ units/second. Besides, we assume there exists no free-riders, i.e., $\min(O_i) > 1$ unit/second, $\forall i \in V$. For the case of $O_i = 1$ unit/second, this reduces the problem to the traditional traveling salesman path problem (TSP), which has been extensively studied in the past [44, 45]. Therefore, we will not focus on this scenario in the proposal.

In the rest of this section, we discuss the details of our algorithm and derive its performance bound.

3.2.2 Centralized Approximation Algorithm

Streaming Backbone Construction

The first step of our algorithm is to construct the virtual streaming backbone. We call it "virtual", because the links on the backbone represent the aggregated inter-cluster streams instead of the actual flows to the representative nodes. Towards that end, we define a metric $C(V')$, called the *residual streaming capacity* for a group of peers $V' \subseteq V$ as:

$$C(V') = \sum_{i \in V'} O_i - (|V'| - 1) \times s. \tag{3.14}$$

The residual streaming capacity represents the contributable bandwidth that a group of peers can supply other groups after satisfying its own streaming demands. Next, we define a threshold γ. The set of peers that are enclosed within the γ radius from peer $v_i \in V$ is denoted as $E_\gamma(v_i, V)$. In addition, we say that v_i γ-covers the peers in $E_\gamma(v_i, V)$ [28].

A parameter t' is chosen, which can be viewed as a "guess" on the optimal P2P streaming delay. A reasonable value of t' should be initialized in the range of $[\max_{v \in V} d_{(S,v)}, |R| \cdot \max_{v \in V} d_{(S,v)}]$. For the given value of t', we set $\gamma = t'/\sqrt{\log n}$. We begin the clustering process from the source node by defining the first cluster $U_1 = E_{3\gamma}(S, W_1^{3\gamma})$ and the first representative node as $u_1 = S$, where $W_1^{3\gamma} = V$, represents the initially un-clustered nodes. For ease of notation, we call the set of peers that are at least γ distance away from existing $i-1$ representatives $u_1, \cdots, u_{i-1}$ as W_i^γ, where $W_i^\gamma = V \setminus \bigcup_{1 \le j \le i-1} E_\gamma(u_j, V)$. We then select a representative node $u_i \in W_i^\gamma$ that γ-covers the peers with the highest residual streaming capacity in $W_i^{3\gamma}$, i.e.:

$$u_i = \text{argmax}_{v \in W_i^\gamma} C(E_\gamma(v, W_i^{3\gamma})). \tag{3.15}$$

Now, we have $U_i = E_{3\gamma}(u_i, W_i^{3\gamma}) \cup E_\gamma(u_i, W_i^\gamma)$. The iteration stops once all the peers in V are 3γ-covered by the existing representatives. Suppose we have k clusters. Then, $W_{k+1}^{3\gamma} = \emptyset$ and $W_i^{3\gamma} \neq \emptyset, \forall 1 \le i \le k$.

Without loss of generality, we reorder the clusters $U_1, \cdots, U_k$ so that the residual streaming capacities of the clusters except that of U_1 are sorted in a non-increasing order, i.e., $C(U_i) \geq C(U_j), \forall 1 < i < j \leq k$. By virtualizing the upload capacity of the representative u_i as $C(U_i)$, we can construct the global streaming mesh of low latency for the backbone representatives. Algorithm 1 describes this procedure. It will be rerun with different values of t' chosen by binary search to achieve the approximated minimum delay.

Corollary 1. *Any peer that is 2γ-covered by u_i must be in the cluster of U_j with $1 \leq j \leq i \leq k$.*

Algorithm 1 APX-MDPS$(G, n, s, t', \{O_i\}, \{I_i\})$: Centralized approximation algorithm APX-MDPS for the MDPS problem

$\gamma = t'/\sqrt{\log n}$
$W_1^{3\gamma} = V$
$U_1 = E_{3\gamma}(S, W_1^{3\gamma})$
$u_1 = S$
$i = 2$
while $W_i^{3\gamma} \neq 0$ **do**
  $u_i = \text{argmax}_{v \in W_i^{\gamma}} C(E_{\gamma}(v, W_i^{3\gamma}))$
  $U_i = E_{3\gamma}(u_i, W_i^{3\gamma}) \cup E_{\gamma}(u_i, W_i^{\gamma})$
  $i = i + 1$
end while
Reorder $U_2, \cdots, U_k$ so that $C(U_i) \geq C(U_j), \forall 2 \leq i < j \leq k$
Construct the backbone mesh by Algorithm 2
Construct the regional mesh by Algorithm 3
Construct the final mesh by Algorithm 4

Algorithm 2 APX-BACKBONE($\{U_i\}, \{u_i\}$): Backbone mesh construction algorithm for the MDPS problem

1: **for** $i = 2$ to k **do**
2: **repeat**
3: $j = \min_{1 \leq n \leq k}\{n : U_n$ has remaining bandwidth$\}$
4: Connect u_i to u_j
5: **until** U_i is fully served
6: **end for**

Regional Streaming Mesh Construction

In this section, we show the steps of creating the regional streaming topology for each cluster. Given a set of clusters, we can identify two types among them by measuring $C(U_i) \geq 0$ or $C(U_i) < 0$. For the cluster with non-negative residual streaming capacities, a mesh spanning the peers of U_i can be constructed by Algorithm 3. For the other type of cluster with negative residual streaming capacities, we can deduce from Lemma 1 that the upload capacities $\sum_{v \in U_i} O_v$ inside the cluster cannot satisfy its internal streaming requirement $(|U_i| - 1) \times s$ and thus need extra streaming connections from external clusters. In such clusters, we will first satisfy the internal peers with the highest upload capacities so that they can timely serve other internal peers. In that way, only the peers with the lowest upload capacities will be left for external connections. Algorithm 3 also describes the method to construct the streaming topology for such clusters.

Algorithm 3 APX-CLUSTER($\{U_i\}, \{u_i\}$): Regional mesh construction algorithm for the MDPS problem

```
1: for i = 1 to k do
2:   for j = 1 to |U_i| do
3:     repeat
4:       if j is the peer with largest uploading capacity then
5:         Hold j for an uplink connection from other cluster
6:       else
7:         x = argmax_{v∈U_i & v has remaining bandwidth} C({v})
8:         if x is fully served then
9:           Connect j to peer x
10:        else
11:          Hold j for external connection
12:        end if
13:      end if
14:    until peer j is fully served or on hold
15:  end for
16: end for
```

Complete Streaming Mesh Construction

To complete the final mesh construction, we replace virtual links between representatives to real inter-cluster connections by Algorithm 4. We then do a binary search over t' to obtain the minimum-delay streaming topology. An outline of the algorithm is described in Algorithm 4. Figure 3.1 illustrates a cluster-based streaming mesh from the source to receivers.

Algorithm 4 APX-COMPLETE($G, \{U_i\}, \{u_i\}$): Complete mesh construction algorithm for the MDPS problem

1: **for** $i = 1$ to k **do**
2: Connect each peer that is held for external connections to the closest available peer in the virtually linked parent cluster.
3: **end for**

Performance Bound

Assume t' is the optimum value. After binary search, it can be approximated within a factor of 2. We now analyze the performance bound of the approximation algorithm, i.e., the approximation factor.

We start from a simple scenario with a streaming rate of $s = 1$ unit/second. In this case, the resulting streaming topology can be expected as a tree structure denoted as T, because each peer will only receive stream from one parent. Let T^* be the optimal tree with the minimum streaming delay, denoted by $T^* = (V, E^*)$ and $E^* \subset E$. Now we partition T^* into clusters $\{U_1^*, \cdots, U_q^*\}$, which are represented by $\{u_1^*, \cdots, u_q^*\}$, respectively. We define two functions: PATH(i, j), which returns true only if there exists a directed path from i to j in T^* denoted as $\langle i, j \rangle$, and HEIGHT(i), which returns the height of node i in tree T^* rooted at S. Let T_B^* be the backbone after partitioning T^*.

Our method to partition T^* is summarized as follows:

1. Let $u_1^* = S$ and U_1^* be the set of descendants that is γ-covered by S, i.e., $U_1^* = \{u : u \in V$, PATH$(S, u) =$ TRUE, and $d_{(S,u)} \leq \gamma\}$.
2. Let W^* be the uncovered peers where $W^* = V \setminus U_1^*$;
3. Select the lowest uncovered node as the next representative u_i^*, i.e., $u_i^* = \text{argmin}_{u \in W^*}\text{HEIGHT}(u)$ and $U_i^* = \{u : u \in W$ and PATH$(u_i^*, u) =$ TRUE and $d_{(S,u)} \leq \gamma\}$;

4. Remove U_i^* from W^*, i.e., $W^* = W^* \setminus U_i^*$;

5. Repeat from Step 3 until all the nodes are covered.

Suppose we have q clusters from T^* after the above steps. Without loss of generality, we then reorder the clusters $U_1^*, \cdots, U_q^*$ so that the residual streaming capacities of them except U_1 are sorted in a non-increasing order, i.e., $C(U_i^*) \geq C(U_j^*), \forall 1 < i < j \leq q$. Moreover, we denote T_B as the backbone constructed from G by Algorithm 1.

Lemma 2. *Suppose U_i^* and U_j intersects, i.e., $U_i^* \cap U_j \neq \emptyset$. Then, there must exist a cluster U_m covering at least one $u \in U_i^* \setminus U_j$ and satisfying $C(U_m) \geq C(U_i^*)$, where $1 \leq m \leq k$.*

Proof. If any peer w in $U_i^* \cap U_j$ is within the γ radius of u_j, i.e., $d_{(w,u_j)} \leq \gamma$, then every peer $u \in U_i^* \setminus U_j$ will be 2γ-covered by u_j. According to Corollary 1, there must exist some cluster U_m covering u , where $1 \leq m \leq j$. Since u_j γ-covers the peers in $U_i^* \cap U_j$, it is easy to see that $C(U_j) \geq C(U_i^*)$. Then we have $C(U_m) \geq C(U_j) \geq C(U_i^*)$.

If $\gamma \leq d_{(w,u_j)} \leq 2\gamma$, then there must exist a cluster U_m, which is the first cluster that covers at least one peer $\in U_i^*$ with $C(U_m) \geq C(U_i^*)$; otherwise, U_i^* will form a cluster itself. Peer $u \in U_i^* \setminus U_j$ may have two possibilities: (1) $d_{(u,u_j)} \leq 2\gamma$, or (2) $2\gamma < d_{(u,u_j)} \leq 3\gamma$. In the case of $d_{(u,u_j)} \leq 2\gamma$, there must exist some cluster U_n covering u with $1 \leq n < j$ according to Corollary 1. Since U_m should have $C(U_m) \geq C(U_n) > C(U_j)$, we deduce $U_m \neq U_j$. In the other case of $2\gamma < d_{(u,u_j)} \leq 3\gamma$, the cluster U_m that covers u must be within the γ radius of u and $C(U_m) \geq C(U_i^*)$; otherwise, U_i^* itself will be more qualified as a cluster than U_m.

If $d_{(w,u_j)} > 2\gamma$, then peer $u \in U_i^* \setminus U_j$ must be within the γ radius of some u_m and $C(U_m) \geq C(U_i^*)$; otherwise, U_i^* will form a cluster itself. Thus, in all cases, the lemma follows. □

Lemma 3. *Compare the residual capacities of T_B and T_B^*. We have:*

$$\sum_{i=1}^{j} C(U_i^*) \leq \sum_{i=1}^{j} C(U_i), \forall 1 \leq j \leq q. \tag{3.16}$$

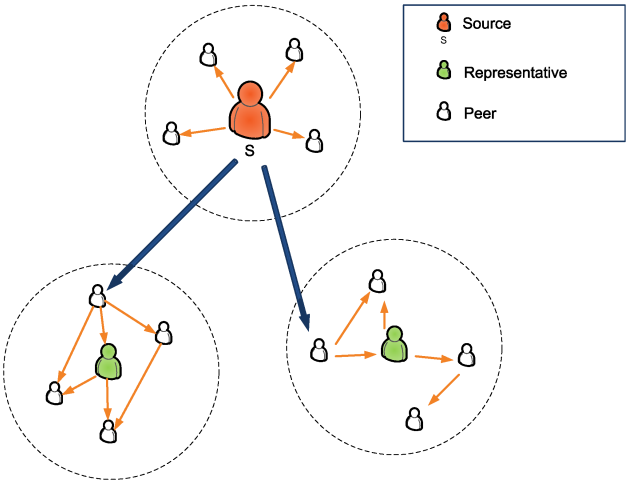

Figure 3.1: Cluster-based streaming mesh.

Proof. We use induction to prove this claim. For $j = 1$, it is obvious that $C(U_1^*) \leq C(U_1)$. Now, we assume that the claim also holds for $j = x$. When $j = x + 1$, we have a new cluster U_{x+1}^*. If $\bigcup_{1 \leq i \leq x} U_i$ contains no peer in U_{x+1}^*, i.e., $U_{x+1}^* \cap \bigcup_{1 \leq i \leq x} U_i = \emptyset$, then we should have $C(U_{x+1}) \geq C(U_{x+1}^*)$ because Algorithm 1 always selects U_{x+1} with the largest residual streaming capacity from the uncovered peers. If $\bigcup_{1 \leq i \leq x} U_i$ contains several peers in U_{x+1}^*, then we have $U_{x+1}^* \cap \bigcup_{1 \leq i \leq x} U_i \neq \emptyset$.

For the sake of contradiction, we assume that $\sum_{i=1}^{x+1} C(U_i^*) > \sum_{i=1}^{x+1} C(U_i)$. If there is any cluster U_n^* with $n \leq x$ satisfying $U_n^* \cap \bigcup_{1 \leq i \leq x} U_i = \emptyset$, then $C(U_{x+1}) \geq C(U_n^*) \geq C(U_{x+1}^*)$. Thus, to make the assumption correct, we have $U_n^* \cap \bigcup_{1 \leq i \leq x} U_i \neq \emptyset, \forall 1 \leq n \leq x$. Therefore, according to Lemma 2, we must have a U_m covering at least one peer in $U_n^* \setminus \bigcup_{1 \leq i \leq x} U_i$ and satisfying $C(U_m) \geq C(U_n^*)$ with $x < m \leq k$. Consequently, we should have $C(U_{x+1}) \geq C(U_m) \geq C(U_n^*) \geq C(U_{x+1}^*)$. Then, $\sum_{i=1}^{x+1} C(U_i^*) \leq \sum_{i=1}^{x+1} C(U_i)$, which contradicts the assumption. The lemma follows. □

Since the sum of the residual capacities is bounded by $\sum_{i \in V} O_i - (n-1) \times s$, we can easily deduce Corollary 2 from Lemma 3:

Corollary 2. *The number of clusters in T is less than that in T^*, i.e., $k \leq q$.*

Lemma 4. *Let H_I denote the height of a tree I. We have $H_{T_B} \leq H_{T_B^*}$.*

Proof. T_B is constructed as a balanced tree. Combining this fact with Lemma 3, which represents a higher out-degree in T_B than that in T_B^*, we can observe that $H_{T_B} \leq H_{T_B^*}$. □

We call the edge connecting different clusters as the *backbone edge* and the edge inside the cluster as the *cluster edge*.

Lemma 5. *Any root-to-leaf path in T_B has at most $O(\sqrt{\log n})$ backbone edges.*

Proof. It follows from Lemma 4 that $H_{T_B} \leq H_{T_B^*}$. From the construction method of T_B^*, we know that any root-to-leaf path in T_B^* has a length, i.e., latency, which is at least $\gamma \cdot H_{T_B^*}$. Since T_B^* has a latency which is at most OPT, it follows that:

$$H_{T_B} \leq H_{T_B^*} \leq \frac{\text{OPT}}{\gamma} = \frac{\text{OPT}}{t'/\sqrt{\log n}} = O(\sqrt{\log n}). \tag{3.17}$$

□

Lemma 6. *Any root-to-leaf path in T has at most $O(\log n)$ cluster edges.*

Proof. Without loss of generality, we can envision T_B is constructed in a breadth-first and left-to-right order, which means representatives on the same height are arranged from left to right in a non-increasing order of their residual streaming capacities. In another word, if u_i is on the left of u_j on the same height in T_B where $0 \leq i, j \leq k$, we have $C(U_i) \geq C(U_j)$. Moreover, if u_i has a lower height in T_B than u_j where $0 \leq i, j \leq k$, we have $C(U_i) \geq C(U_j)$.

Denote $T(U)$ as the tree that is constructed inside the cluster U. Let Y be the rightmost root-to-leaf path in T_B, denoted as $Y = \langle u_1, u_{y_1}, \cdots, u_{y_b} \rangle$, i.e., Y is formed from root to leaf by representatives $u_1, u_{y_1}, \cdots, u_{y_b}$. For any root-to-leaf path X, where $X = \langle u_1, u_{x_1}, \cdots, u_{x_a} \rangle$, we denote the number of cluster edges in X as ζ_X. Since T_B is constructed as a balanced tree, we can deduct $b \leq a \leq b + 1$.

Because Y is the rightmost root-to-leaf path in T_B, it follows that

$$\begin{aligned}\zeta_X &= H_{U_1} + \sum_{i=1}^{a} H_{T(U_{x_i})} \\ &= H_{U_1} + H_{U_{x_1}} + \sum_{i=2}^{a} H_{T(U_{x_i})} \\ &\leq H_{U_1} + H_{U_{x_1}} + \sum_{i=1}^{b} H_{T(U_{y_i})}.\end{aligned}$$

Recall the previous assumption that $\min(O_i) \geq 2, \forall i \in V$. For the case of $b \leq 1$, it is easy to deduct $\zeta_X \leq 3\log_2 n = O(\log n)$. Otherwise, we can carry out

$$\begin{aligned}\zeta_X &\leq H_{U_1} + H_{U_{x_1}} + \sum_{i=1}^{b} \log_2 \left(C\left(U_{y_i}\right) - 1\right) \\ &= H_{U_1} + H_{U_{x_1}} + \log_2 \left(\prod_{i=1}^{b} \left(C\left(U_{y_i}\right) - 1\right)\right). \end{aligned} \tag{3.18}$$

In addition, the number of the clusters, i.e. k, is bounded by n. Thus, we can deduct

$$\begin{aligned}n \geq k \geq & 1 + C\left(U_1\right) + C\left(U_1\right) \cdot C\left(U_{y_1}\right) + \\ & \cdots + C\left(U_1\right) \cdot \prod_{i=1}^{b-1} C\left(U_{y_i}\right) \\ & = 1 + C\left(U_1\right) \cdot \left(1 + \sum_{i=1}^{b-1} \prod_{j=1}^{i} C\left(U_{y_i}\right)\right).\end{aligned}$$

Thus, we have

$$C\left(U_1\right) \cdot \prod_{i=1}^{b-1} C\left(U_{y_i}\right) \leq n. \tag{3.19}$$

Replacing Equation (3.19) into (3.18), we have

$$\begin{aligned}
\zeta_X &\leq H_{U_1} + H_{U_{x_1}} + \log_2 \left(\prod_{i=1}^{b} C\left(U_{y_i}\right) \right) \\
&= H_{U_1} + H_{U_{x_1}} + \log_2 \left(C\left(U_1\right) \cdot \prod_{i=1}^{b-1} C\left(U_{y_i}\right) \right) \\
&\quad + \log_2 \left(C\left(U_{y_b}\right) / C\left(U_1\right) \right) \\
&\leq H_{U_1} + H_{U_{x_1}} + \log_2 n + \log_2 \left(C\left(U_{y_b}\right) / C\left(U_1\right) \right) \\
&\leq 4 \log_2 n \\
&= O(\log n). \qquad (3.20)
\end{aligned}$$

Thus, the lemma follows. □

Theorem 2. *Let OPT be the minimum P2P streaming delay from the source host S to receivers R in T. The streaming delay of the solution produced by Algorithm APX-MDPS is at most $O(\sqrt{\log n}) \cdot OPT$.*

Proof. It follows from Lemma 5 that any root-to-leaf path has at most a latency of $O(\sqrt{\log n}) \cdot$ OPT arising from the backbone edges on the path. In addition, short edges in T have a latency that is no more than $6\gamma = 6t'/\sqrt{\log n}$. From Lemma 6, we know that any root-to-leaf path has at most a latency of $6\gamma \cdot O(\log n) = O(\sqrt{\log n}) \cdot$ OPT caused by cluster edges on the path. T is constructed from the backbone edges and the cluster edges. The theorem follows. □

Now, let us look at the problem when $s > 1$ units/second.

Theorem 3. *Let OPT be the minimum P2P streaming delay from the source host S to receivers R in G. The streaming delay of the solution that Algorithm APX-MDPS returns is at most $O(\sqrt{\log n}) \cdot OPT$.*

Proof. When $s > 1$ unit/second, the final streaming topology will be a mesh, which can be envisioned as a combination of multiple trees constructed by fractional streams. To prove the

previous bound also holds here, we first normalize all flow rates and capacities by s. Then, the correctness of Lemmas 1-5 is obvious. For Lemma 6, we start justifying its correctness from Equation (3.18). Recall the assumption that at least half of the nodes have $O_i \geq 2s$. Because of the balanced topology in connection, we can prove that any node whose $O_i < 2s$ will always lay on the bottom in its cluster. (Due to space limitations, we do not provide detailed proof of this claim.) As a result, we can carry out

$$\begin{aligned}
\zeta_X &\leq H_{U_1} + H_{U_{x_1}} + \sum_{i=1}^{b} \log_2 \left(C\left(U_{y_i}\right) \times \frac{2s}{\min(O_i)} - 1 \right) \\
&\leq H_{U_1} + H_{U_{x_1}} + \sum_{i=1}^{b} \log_2 \left(C\left(U_{y_i}\right) \times \frac{2s}{\min(O_i)} \right) \\
&= H_{U_1} + H_{U_{x_1}} + \log_2 \left(\prod_{i=1}^{b} C\left(U_{y_i}\right) \right) + b \cdot \log_2 \frac{2s}{\min(O_i)} \\
&= H_{U_1} + H_{U_{x_1}} + \log_2 \left(\prod_{i=1}^{b} C\left(U_{y_i}\right) \right) + \log_2 n \times \log_2 s. \qquad (3.21)
\end{aligned}$$

Similar to the deduction of Equation (3.20), we have

$$\begin{aligned}
\zeta_X &\leq (4 + \log_2 s) \cdot \log_2 n \\
&= O(\log n). \qquad (3.22)
\end{aligned}$$

Thus, the theorem follows.

□

3.2.3 Distributed Algorithm

The centralized algorithm described in Section 3.2.2 approximately solves the minimum-delay P2P streaming problem. In a practical setting, however, we may not have a central

server that can provide a global computation resource which is implicitly assumed by the algorithm. Thus, to increase the scalability and reliability, we extend our algorithm to a distributed version, which can be well adopted and improved as a practical P2P protocol. For the ease of presentation, we assume that all the representatives have sufficient resources to coordinate the peers. In the actual deployment, we define a *cluster leader* to take charge, which is the peer with the most computing and bandwidth resources in the cluster.

Peer Join

In order to join the clusters, a newly arrived peer i will first contact a *rendezvous point* (RP), which caches a list of existing representative peers. The rendezvous point then sends back a random list of representatives that are approximately nearby the newcomer and the updated parameter of γ. Peer i will then check the latency and the residual streaming capacity with each representative u_j. If there exists an u_j within γ latency of i, peer i will send a request to join the closest cluster; otherwise, peer i will tentatively join the closest cluster, but meanwhile measure the feasibility to build a new cluster.

In the attempt to organize a new cluster, peer i will contact a set of nearby representatives and will retrieve a list of peers that are between γ and 3γ-away from each representative u_j. Next, peer i will pick the peers within its γ radius and activate the new clustering process on these peers. In the process, they start exchanging the latency and capacity information with each other and find the center peer, i.e., the representative, which has the maximum residual streaming capacities for the attempted new cluster. If the residual streaming capacity of the candidate cluster is less than the existing cluster within 3γ radius of j, the new clustering process will be terminated; otherwise, the new representative will request peers within its γ latency to join the new cluster.

Finally, the new representative will request peers within its γ latency to join the new cluster. After a peer joins a cluster, its representative will allocate the parents and children for it. Once the new peer receives this information, it will initiate the stream with those

parents and children directly.

Peer Departure

The departures and failures of peers may lead to interrupted playback at the remaining receivers. If any peer departs from the cluster, it will inform its representative and request re-allocating the bandwidth for its downstream peers. To handle this problem of failure, each peer will buffer for a short period when streaming and always keep an eye for the back-up peers during streaming. If failure does happen, the downstream peer will utilize those time of buffering to connect to the backup peers tentatively and then request a stream re-allocation to the representative. If it is the representative that fails or leaves, the affected peers will use the buffering time to activate a new round of clustering within γ radius.

Stream Coordination and Dynamic Adaptation

The topology maintenance and stream allocation are mostly coordinated by the representatives. They will assign a new peer to connect to the peer with the most capacity in its list. If the residual streaming capacity is non-negative, they will always maintain an intra-cluster streaming by utilizing the idle bandwidth. If the residual streaming capacity is negative, they will coordinate with the rendezvous point and inform the unserved peers to stream from the cluster with the most residual capacity. When a new cluster is established, the same steps are followed to import a complete stream to the cluster. If the γ needs to be tuned or optimized, the clustering process will be initiated by the rendezvous point and run in the background without interrupting the existing streams. All connections are updated until the background computation is completed.

In a distributed environment, peers may join and leave randomly. Thus, peers with high bandwidth resources may arrive late and as a result, connect far from the source. Cluster representative and rendezvous point will be responsible for monitoring this scenario at the

cluster level and backbone level, respectively. Once they detect this scenario, they initiate new rounds of stream allocation in the background and update the connections until they reserve the bandwidth.

3.3 Simulation Study

In this section, we describe the results of our simulation study. We simulate a live streaming session of 300 Kbps from a source with 10 Mbps upload capacity. From previous studies, we know that network bandwidth exhibits rich diversity [14, 41]. Based on this, we set the upload capacity among peers as shown in Table 3.1. In this simulation study, we compare our algorithm with two other recently proposed algorithms: a heuristic approach [14] and a LP-based approach [22]. The heuristic in [14] is the first algorithm that focuses on reducing the maximum end-to-end delay on mesh streaming, where peers select their parents based on the metric of link capacity divided by the communication delay [14]. The LP-based approach in [22] applies several linear programming techniques to obtain an optimal average delay, such as Lagrangian relaxation and subgradient algorithm. Please note, to reduce computational costs, [22] restricts the potential connections for each peer. This actually lowers down the performance compared with real LP-based solution. However, for the easy of presentation, we still call it LP-based solution in the rest of this proposal.

To evaluate the algorithm performance, we define four metrics, including *average end-*

Upload Capacity	Percentage of Peers
200 Kbps	30%
1.0 Mbps	50%
2.0 Mbps	15%
10.0 Mbps	5%

Table 3.1: Upload Capacity Distribution

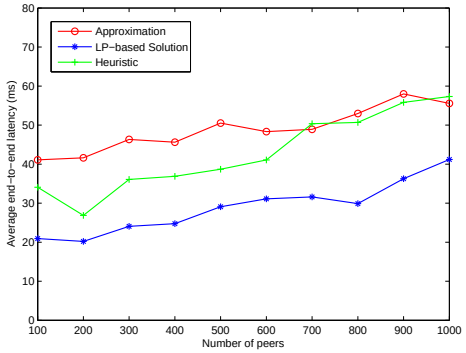

Figure 3.2: Average end-to-end latency

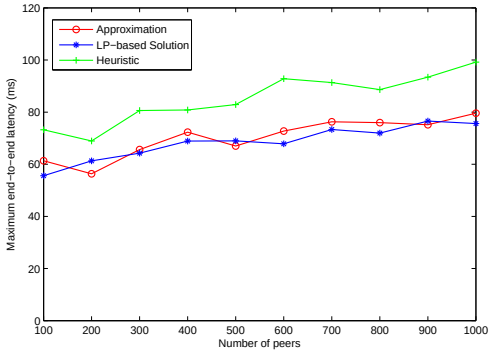

Figure 3.3: Maximum end-to-end latency

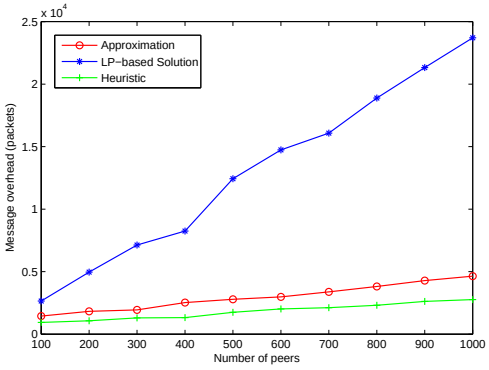

Figure 3.4: Message overhead

to-end delay, *maximum end-to-end delay*, and *message overhead*.

The average end-to-end delay is defined as the average latency from the source to all receivers. Figure 3.2 illustrates the results from our simulation experiments. It shows that the LP-based approach generally achieves a lower average delay than the other two approaches. This is reasonable, since the LP-based approach is designed to minimize the average delay. It is also interesting to observe that the heuristic algorithm exhibits lower average delay than the approximation algorithm when the network size is relatively small. When the network size approaches 700 nodes, the approximation algorithm yields an average latency that is close to that of the heuristic approach. This indicates that the approximation algorithm has a smaller growth rate with respect to increase in network size, implying that the algorithm is scalable for large network sizes.

We also measure the maximum delay, which is the worst-case end-to-end delay observed in the simulation experiments. Note that this is our primary design objective. Figure 3.3 shows the maximum delay of the algorithms. It is apparent that the worst-case performance of our algorithm is close to that of the LP-based solution and outperforms the heuristic. This low worst-case delay indicates that our algorithm ensures good streaming performance with an approximation bound.

Figure 3.4 shows the message overhead of the algorithms, measured in number of packets, during mesh construction and maintenance. Although a minimum delay is desirable, a large message overhead will challenge practical deployment of the underlying algorithm. As we can observe, the LP-based solution generates a huge number of overhead packets during mesh construction. In addition, its overhead significantly increases with the network size. This is mainly resulting from the computational message exchange. In contrast, the heuristic and the approximation algorithm both have much less message overhead and slow growth rate, as the network size increases. The major overhead of our algorithm occurs in clustering. From the simulation results, we observe that this overhead is slightly higher than that of the heuristic algorithm.

Thus, our simulation results reveal the effectiveness of our algorithm, in terms of ensuring a worse-case delay with high scalability.

Chapter 4

On Reducing Churn-induced Delay

In this chapter, we describe the problem on reducing churn-induced delay in P2P streaming, including channel-switching delay and recovery delay. First, we formulate the minimum churn-induced-delay problem. Then we present our proposed agent-based solution and justify its performance by the queueing theory model we develop for P2P scenario specifically. The results of numerical studies validate the effective performance of our scheme at the end of this chapter.

4.1 Problem Formulation and Methodology

For churns in P2P live streaming systems, we can generally classify them into two categories: *peer churn* and *channel churn*. Peer churn arises from peer arrival and departure [6], and channel churn comes from channel switching [9, 46]. In this section, we first formulate our problem for two types of churn-induced delay respectively and then describe the scheme to reduce it.

4.1.1 On Reducing Delay from Channel Churn

We consider a peer-to-peer streaming system with Q channels, where each channel is associated with a distinctive streaming overlay. We denote the set of channels as $\mathbb{C}$. Given a channel $C_i \in \mathbb{C}$, its streaming overlay can be modeled as $G_i = (V_i, E_i)$, where V_i is the set of vertices representing peer nodes on this overlay, and E_i is the set of overlay edges representing directed overlay streaming links. Each channel $C_i \in \mathbb{C}$ is considered as a streaming session, originating from a single source node S_i to a set of receivers R_i, where $\mathbb{C} = \bigcup_{i=1}^{Q} C_i$ and $V_i = \{S_i\} \cup R_i$. Suppose S_i streams data at a constant streaming rate of s_i units/second. If a peer p viewing channel C_i receives the aggregated stream at s_i units/second from its parents, we call peer p as *fully served* [14, 36]. We assume that a fully served peer can smoothly play back the streaming content at its original rate of s_i units/second [14].

To understand the switching delay problem, we first describe a general channel switching protocol without switching delay optimization [5,47]. Once a peer p switches to a new channel C_i, it will initiate a contact with a bootstrap server (BS), which is a server maintaining a partial list L of current peers on the new overlay [5, 48]. After procedures of authentication and authorization for new channel connection [47], the BS will send p a list of random peers L_R as the connection candidates for p, where $L_R \subset L$ [47]. To distinguish with the bootstrapping peers which will be mentioned later in our scheme, we call here the list of peers retrieved from BS as *ordinary peers*. Then p will request streaming from L_R. Active peers in L_R will response with their IP address and data chunk information. The sum of requested streaming rates from L_R should be equal to s_i. If the cumulative available bandwidth from responding peers in L_R satisfies streaming rate and the cumulative chunk is complete, p will attempt the streaming process with them; otherwise, p will request a new list of connection candidates from peers of L_R, exchange information with new list of peers, and retry connections. If the above bootstrap procedure is successful, an overlay mesh for channel C_i will be updated, and subsequently p will receive the channel content from its upstream peers. Clearly, the expected channel switching delay for peer p, denoted as $\overline{d}_{sw}$,

can be expressed by

$$\overline{d}_{sw} = \overline{d}_A + \overline{d}_{ls} + \overline{d}_p + \overline{d}_{sc}, \tag{4.1}$$

where $\overline{d}_A$ is the expected time cost in authorization and authentication before watching a channel, $\overline{d}_{ls}$ is the expected delay to retrieve a peer list of the target channel from the bootstrap server, $\overline{d}_p$ is the expected time spent in initializing the contact with those peers in the list, and $\overline{d}_{sc}$ includes the delay in exchanging data chunk information with responding peers in the list, the time to schedule downloading different chunks from specific peers and the communication delay to receive the chunk data. We now define the problem formally:

Definition 1. ***Minimum Channel Switching Delay Problem (MCSD problem):*** *Given the delay constraint of each type in channel switching process, the MCSD problem is to devise an overlay switching scheme which minimizes the average channel switching delay with each receiver successfully switching to the new channel.Thus, the problem can be transformed to the following optimization problem:*

Measurement studies have revealed the channel switching delay mainly arises from the bootstrap process [5]. Traditional channel switching protocol arranges the threads of bootstrap and streaming in sequence. We can see those two threads are independent with each other, which leads to the feasibility of parallelizing the threads of boot strap and streaming. In light of that, we propose a more time-efficient protocol to leverage such characteristic. In an intuitive scheme, peers can proactively start bootstrapping in other channels, including getting authorization and retrieving list of peers in the channel while viewing current channel, so it will be in the readiness to switch channel. As we know, a typical live streaming application, such as PPLive and UUSee, may accommodate hundreds of channels [5, 49]. It is not practical for each peer to personally maintain bootstrapping in all the other channels due to the expensive message overhead.

Towards a feasible solution for this problem, our protocol suggests a distributed agent-based bootstrap, where each channel will maintain a set of agents responsible for bootstrap-

ping in the channel for external peers. Agents in channel C_i, denoted as $\mathbb{A}_i$, are selected peers with predicated long lifetime, more communication and storage capabilities in the overlay. They collect the bootstrap information about available peers in the channel with periodical update, and then exchange such information with agents in other channels. It is worth mentioning when updating, only changed information will be collected and distributed to other agents. On the background of current streaming, peer p, once joining the network, will register at one of the agents $a \in \mathbb{A}_i$ in its current channel C_i and send a the request for proactive bootstrap to other channels $\bigcup_j C_j$, where $j \neq i$. Agent a will buffer and periodically forward such requests in a package to agents in other channels. Those agents in other channel will authenticate and authorize p's join request. Once request is approved, they notify a. Given the bootstrapping information stored at a as $\mathbb{I}_a$, a will then send p a list of peers $\bigcup_j D_p^j$ about other channels, where $D_p^j \subset \mathbb{I}_a$ and j corresponds to channel C_j. Since then, a will update such bootstrapping information if change happens. Bootstrapping data sent to p will only contain a partial set of peers stored in a'slist for each channel, as long as the available bandwidth in each channel can fully serve p's streaming. In addition, agent will distill the bootstrapping peers with superior bandwidth and lifetime expectation from ordinary peers in its list to quickly serve the viewer in the initial period of streaming. Moreover, agent will pre-schedule the downloading plan about data chunk partition and streaming for future viewers, and coordinate with each other to build the preventive connections for peers represented by them. We call such distilled peers in the bootstrapping data for peer p as *bootstrapping peers*,denoted as B_p^i for channel C_i. When p switches to a new channel C_j, it will call a service to the agent in other channel and simultaneously request streaming from ordinary peers in D_p^j. Agent will arrange forwarding the service request to the bootstrapping peers in its management for a quick assistance in the initial streaming period. After bootstrapping peers starts streaming, they will keep looking for other ordinary peers to take over their streaming job to p. So agent can recycle back the bootstrapping peers. By way of this proactive bootstrap, channel switching delay will be reduced.

To save the massage overhead for updating data on agents, bootstrapping peers in each

channel can work only for the initial streaming period T and during the time hand over the streaming to other ordinary peers they found, called *handover peers*. It is reasonable to evaluate T as the expected time that bootstrapping peers cost to find handover peers. Because bootstrapping peers can be reused in this way, agents will not need to update the bootstrapping peers as long as they are still in the channel. To reduce the influence from channel switching of bootstrapping peers, strategy of view-upload decoupling (VUD) can be applied with our scheme, where what a peer uploads is independent of what it views [9]. Similar to agent selection, bootstrapping peers should have predicted long lifetime and more bandwidth resources. We will discuss the properties of bootstrapping peers and agents in Section 4.2.

Furthermore, given a peer p that is not new in the network, i.e., p has been viewing some channel in the network, when switching channel it will only register at the new agent a but not send bootstrap request since it has done this previously. Bootstrapping data has the time stamp. On p's registration, a will compare the time stamp t_a on its bootstrapping data with t_p that is stored in p. If $t_a > t_p$, a will update the bootstrapping data to p.

4.1.2 On Reducing Delay from Peer Churn

In this section, we discuss the protocol to reduce the delay from another type of churn, i.e., peer churn. Peers can randomly join or leave the overlay of current channel. For most P2P live streaming applications, peer departures have a greater detrimental effect on the delay than new peers arrivals. Peer departures may lead to streaming interruption to their downstream peers. In the proposal, we focus on reducing the delay from peer departure.

We call the departure-induced delay as *recovery delay*, which occurs when downstream peers restore their streaming connections. We denote the recovery delay delay as $\overline{d}_{re}$. It can be carried out that

$$\overline{d}_{re} = \overline{d}_p + \overline{d}_{sc}. \tag{4.2}$$

Comparing with switching delay in Equation (4.1), we can notice a recovery delay has no $\overline{d}_A$ and $\overline{d}_{ls}$ since the peer has already been authorized and owns the peer list in current channel. We now define the problem:

Definition 2. ***Minimum Departure-induced Delay Problem (MDD problem):*** *The MDD problem is to devise an overlay resilience scheme which minimizes the recovery delay during peer departure.*

Similar to MSCD problem, we can observe the feasibility of parallelizing the threads of streaming and recovery in this problem. Thus, we can reduce the delay by keeping a proactive bootstrap to the overlay of current channel. This can be accomplished by a simple modification on the protocol described in Section 4.1.1. Specifically, given an agent a for peer p that is viewing channel C_i, $\mathbb{I}_a$ stored on a will extend the coverage of its bootstrapping data to the current channel C_i. Likewise, D_p^i is stored and updated in peer p as well.

4.1.3 Agent

Agent will exchange information with other. A resilient way to realize that is gossip based method. Due to the page limit, we will not focus on how to implement an optimal gossip plan. In addition, agent may leave the network, so peers may not reach the agent when they call for a service to the agent. Thus, heart-beat signal will send among agents to timely monitor the existence of the agent. In the section of numerical studies, we will detail the setting of heart-beat signal frequency. Once an agent departure is detected, neighboring agents of this agent will initialize a process to select new agent. Neighboring agents exchange the copies of bootstrapping information. So when they select new agent, they will put back such information in the left agent to the new agent.

4.2 Modeling and Analysis

In this section, we formally model and analyze our scheme described in the previous section.

We call the newly arrived peers who did not view any channels *new peers* to the network. For peers that have been viewing channel content, we call them *existing peers*. We model the arrival of new peers to a channel by a Poisson process. In detail, suppose the expected number of new peer arrivals in one time unit is λ_n, then the probability that there are exactly k arrivals of new peers in t time units is equal to

$$f(k; \lambda_n t) = \frac{(\lambda_n t)^k e^{-\lambda_n t}}{k!}.$$

Likewise, we model arrivals of existing peers from any other channel by Poisson process. Suppose the expected number of existing peers switching from C_i to C_j in one time unit is $\lambda_{i,j}$, the probability that there are exactly $k_{i,j}$ of such arrivals in time units is expressed by

$$f(k_{i,j}; \lambda_{i,j} t) = \frac{(\lambda_{i,j} t)^{k_{i,j}} e^{-\lambda_{i,j} t}}{k_{i,j}!}.$$

Theorem 4. *Given the expected lifetime of peers on channel C_j as $1/\mu_j$, the expected number of viewers on C_j, denoted as $\overline{N}_j$, is $(\sum_i \lambda_{i,j} + \lambda_n)/\mu_j$ in steady state, where $i \neq j$.*

Proof. It is known the sum of two independent Poisson variables still follow the Poisson distribution. Mathematically, given $X_1 \sim \text{Poisson}(\lambda_1)$ and $X_2 \sim \text{Poisson}(\lambda_2)$, $Y \sim \text{Poisson}(\lambda_1 + \lambda_2)$ where $Y = X_1 + X_2$. In light of this, we can iteratively deduct the sum of multiple finite Poisson variables also follows a Poisson distribution. Thus, the arrival rates on Channel C_j follow $\text{Poisson}(\sum_i \lambda_{i,j} + \lambda_n)$.

The P2P streaming system is modeled as an $M/G/\infty$ system here. According to the queueing theory, the expected number of viewers can be carried out by

$$\overline{N}_j = (\sum_i \lambda_{i,j} + \lambda_n)/\mu_j, \tag{4.3}$$

where $i \neq j$.

Thus, the theorem follows. □

Suppose the lifetime of bootstrapping peers, denoted as L_s, follows an exponential distribution, whose probability density function (PDF) can be expressed by

$$f_{L_s}(t;\gamma) = \begin{cases} \gamma e^{-\gamma t} & \text{if } t \geq 0, \\ 0 & \text{if } t < 0, \end{cases}$$

where $1/\gamma$ is the expected lifetime of bootstrapping peers.

As we know, agent will collect the bootstrapping information in its channel and periodically update it. To simplify the problem, we assume the update frequencies on all agents are the same as F. Define the probability that a bootstrapping peer is in the network when it is called to serve other peers as *availability*.

Theorem 5. *The expected availability of bootstrapping peer is* $(1 - e^{-\gamma/F})F/\gamma$.

Proof. From the lifetime distribution of bootstrapping peers, we can carry out the probability that it will stay for at least time t by

$$\begin{aligned} \Pr\{L_s \geq t\} &= 1 - F_{L_s}(t;\gamma) \\ &= e^{-\gamma t}, \end{aligned} \tag{4.4}$$

where F_{L_s} is the cumulative distribution function (CDF) of L_s.

As we know, the exponential distribution is memoryless, which means its conditional probability obeys

$$\Pr\{L_s \geq T_R + t | L_s > T_R\} = \Pr\{L_s \geq t\}. \tag{4.5}$$

Let T_R be update time when an agent is checking if the bootstrapping peer is still in the network. If some bootstrapping peers has left the network, agent will fill in new ones. According to Equation (4.5), once a bootstrapping peer is in the network on the update point, the probability that it will stay for at least time t will be the same as those newly joined bootstrapping peers. Thus, we can apply the same Equation (4.4) for all bootstrapping peers.

It is obvious that the time of a service call for a bootstrapping peer, i.e., t in Equation (4.4), is uniformly distributed on the time between $[T_R, T_R + T_a]$ where $T_a = 1/F$. Accordingly, the expected availability of a bootstrapping peer on a service call, denoted as $\overline{A}_v$, is

$$\begin{aligned}
\overline{A}_v &= \frac{1}{T_a}\int_{T_R}^{T_R+T_a} \Pr\{L_s \geq T_R + t | L_s > T_R\}\mathrm{d}t \\
&= \frac{1}{T_a}\int_{0}^{T_a} \Pr\{L_s \geq t\}\mathrm{d}t \\
&= \frac{1}{T_a}\int_{0}^{T_a} e^{-\gamma t}\mathrm{d}t \\
&= \frac{1 - e^{-\gamma T_a}}{\gamma T_a} \\
&= \frac{(1 - e^{-\gamma/F})F}{\gamma}.
\end{aligned}$$

Thus, the theorem follows.

□

Corollary 3. *Suppose the* $1/\gamma_a$ *is the expected lifetime of agents and* F_a *is the frequency of heart-beat signal to check if the agent is still in the network. The expected availability of an agent* $\overline{A}_a$*is* $(1 - e^{-\gamma_a/F_a})F_a/\gamma_a$.

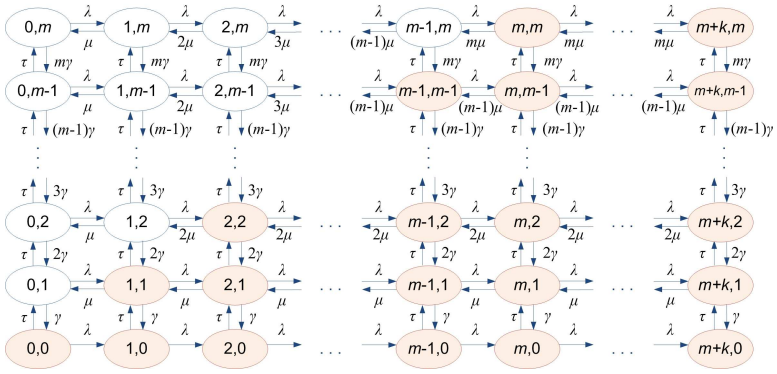

Figure 4.1: M/M/m/$m+k$ model

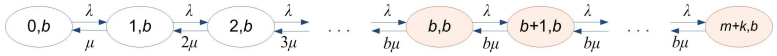

Figure 4.2: Isolated model on b^{th} level

Suppose that agent can store at most m bootstrapping peers and serve up to $m+k$ new peer requests, where m and k are both non-negative integers. To facilitate the modeling of our scheme, we make the following assumptions: 1) the arrivals of new viewers to the agent follow a Poisson process, denoted as Poisson(λ) where λ is the expected arrival rate; 2) the service time of a bootstrapping peer, i.e., the time it takes to hand over the current streaming to other peers, follows an exponential distribution, denoted as Exponential(μ) where $1/\mu$ is the expected service time; 3) the arrivals of new bootstrapping peers to the agent follow Poisson(τ) where τ is the expected arrival rate.

Accordingly, we can model the bootstrapping system on each agent by an M/M/m/$m+k$ queue with bootstrapping peer failure and repair. Figure 4.1 illustrates our multi-level model, where color-shaded states means available bootstrapping peers are all busy. Given a state (a, b), a represents the number of viewers that are served or waiting to be served by bootstrapping peers, and b describes the number of available bootstrapping peers. To simplify the modeling, we assume a bootstrapping peer can only serve one peer at a time, i.e., $J = 1$. For other numbers of J, this model can also work well after simple modifications on b and state transition parameters.

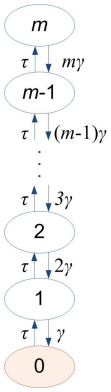

Figure 4.3: Single-column model

We sequentialize the levels, i.e. rows, in M/M/m/$m+k$ from bottom to top, beginning with 0. Thereby, b^{th} *level* consists of states that have b available bootstrapping peers. To obtain the steady-state probabilities, we can first isolate out each level in state graph as a single 1-level model, analyze it for the steady-state probabilities, and then replace each row by a single state in the original model, analyze this single-column model and carry out the joint probabilities from both models. Figure 4.2 shows the isolated model on b^{th} level. After replacing each row by a single state in M/M/m/$m+k$ model, we can obtain Figure 4.3, where each state (b) represents the number of available bootstrapping peers.

Theorem 6. *The probability that a working bootstrapping peer is immediately available for a service call is*

$$\sum_{b=1}^{m}\sum_{n=0}^{b-1}\frac{(\lambda/\mu)^n(\tau/\gamma)^b}{n!b![\sum_{i=0}^{m}\frac{1}{i!}(\frac{\tau}{\gamma})^i][\sum_{i=0}^{b}\frac{1}{i!}(\frac{\lambda}{\mu})^i+\sum_{i=b+1}^{m+k}\frac{1}{b^{i-b}b!}(\frac{\lambda}{\mu})^i]}.$$

Proof. First, we analyze the isolated 1-level model on each row. On the b^{th} level, the probability that a working bootstrapping peer is immediately available once a viewer calls for a service is the sum of probabilities on unshaded states, including states $(0,b),(1,b),...\text{and}(b,b)$. To solve the steady-state probabilities of the b^{th}-level model, we use the local balance on each state, i.e. the flow of probability into a state equals the flow out of the same state. Let the probability on state (n,b) be $\Pr(n,b)$. Thus, we have the following table of state equilibriums.

State	Equilibrium
$(0, b)$	$\lambda \Pr(0, b) = \mu \Pr(1, b)$
$(1, b)$	$\lambda \Pr(1, b) = 2\mu \Pr(2, b)$
$\vdots$	$\vdots$
(b, b)	$\lambda \Pr(b, b) = b\mu \Pr(b+1, b)$
$(b+1, b)$	$\lambda \Pr(b+1, b) = b\mu \Pr(b+2, b)$
$\vdots$	$\vdots$
$(m+k-1, b)$	$\lambda \Pr(m+k-1, b) = b\mu \Pr(m+k, b)$

By solving the equilibriums in the table, we can carry out the probability on state (n, b) by

$$\Pr(n, b) = \begin{cases} \frac{1}{n!}(\frac{\lambda}{\mu})^n \Pr(0, b) & \text{if } n \leq b, \\ \frac{1}{b^{n-b}b!}(\frac{\lambda}{\mu})^n \Pr(0, b) & \text{if } n > b. \end{cases} \tag{4.6}$$

Second, we replace each row by a single state in the original M/M/m/$m+k$ model, and analyze this single-column model. Obviously, the model in Figure 4.3 is a M/M/m/m queue. Applying the same method, we can obtain

$$\Pr(0) = \frac{1}{\sum_{i=0}^{m} \frac{1}{i!}(\frac{\tau}{\gamma})^i};$$
$$\Pr(b) = \frac{1}{b!}(\frac{\tau}{\gamma})^b \Pr(0).$$

Next, we apply normalization equation, i.e. $\sum_{i=0}^{m+k} \Pr(i, b) = \Pr(b)$. Therefore, the following can be carried out

$$\begin{aligned}\Pr(0,b) &= \frac{\Pr(b)}{\sum_{i=0}^{b}\frac{1}{i!}(\frac{\lambda}{\mu})^i + \sum_{i=b+1}^{m+k}\frac{1}{b^{i-b}b!}(\frac{\lambda}{\mu})^i} \\ &= \frac{(\frac{\tau}{\gamma})^b \Pr(0)}{b![\sum_{i=0}^{b}\frac{1}{i!}(\frac{\lambda}{\mu})^i + \sum_{i=b+1}^{m+k}\frac{1}{b^{i-b}b!}(\frac{\lambda}{\mu})^i]} \\ &= \frac{(\tau/\gamma)^b}{b![\sum_{i=0}^{m}\frac{1}{i!}(\frac{\tau}{\gamma})^i][\sum_{i=0}^{b}\frac{1}{i!}(\frac{\lambda}{\mu})^i + \sum_{i=b+1}^{m+k}\frac{1}{b^{i-b}b!}(\frac{\lambda}{\mu})^i]}.\end{aligned} \tag{4.7}$$

So in general, the probability that a working bootstrapping peer is immediately available for service, denoted as A_s, can be obtained by

$$\begin{aligned}A_s &= \sum_{b=1}^{m}\sum_{n=0}^{b-1}\Pr(n,b) \\ &= \sum_{b=1}^{m}\sum_{n=0}^{b-1}\frac{(\lambda/\mu)^n(\tau/\gamma)^b}{n!b![\sum_{i=0}^{m}\frac{1}{i!}(\frac{\tau}{\gamma})^i][\sum_{i=0}^{b}\frac{1}{i!}(\frac{\lambda}{\mu})^i + \sum_{i=b+1}^{m+k}\frac{1}{b^{i-b}b!}(\frac{\lambda}{\mu})^i]}.\end{aligned}$$

Thus, the theorem follows.

□

Theorem 7. *The probability that a viewer will be rejected when the agent already has $m+k$ service requests is*

$$\sum_{b=0}^{m}\frac{(\lambda/\mu)^{m+k}(\tau/\gamma)^b}{b^{m+k-b}(b!)^2[\sum_{i=0}^{m}\frac{1}{i!}(\frac{\tau}{\gamma})^i][\sum_{i=0}^{b}\frac{1}{i!}(\frac{\lambda}{\mu})^i + \sum_{i=b+1}^{m+k}\frac{1}{b^{i-b}b!}(\frac{\lambda}{\mu})^i]}.$$

Proof. The probability that a viewer will be rejected, denoted as $\Pr(R)$, is the sum of probabilities on the right-most states, i.e.,

$$\Pr(R) = \sum_{b=0}^{m}\Pr(m+k,b).$$

Applying the results in Equation (4.6) and (4.7), we can have

$$\Pr(R) = \sum_{b=0}^{m} \frac{(\lambda/\mu)^{m+k}(\tau/\gamma)^b}{b^{m+k-b}(b!)^2[\sum_{i=0}^{m}\frac{1}{i!}(\frac{\tau}{\gamma})^i][\sum_{i=0}^{b}\frac{1}{i!}(\frac{\lambda}{\mu})^i + \sum_{i=b+1}^{m+k}\frac{1}{b^{i-b}b!}(\frac{\lambda}{\mu})^i]}.$$

□

Theorem 8. *The probability that all bootstrapping peers in an agent has left the network is*

$$\frac{1}{\sum_{i=0}^{m}\frac{1}{i!}(\frac{\tau}{\gamma})^i}$$

Proof. The probability that all bootstrapping peers are offline, denoted as $\Pr(L)$, is the sum of probabilities on the bottom states, i.e.,

$$\begin{aligned}\Pr(L) &= \sum_{n=0}^{m+k} \Pr(n, 0) \\ &= \Pr(0) \\ &= \frac{1}{\sum_{i=0}^{m}\frac{1}{i!}(\frac{\tau}{\gamma})^i}.\end{aligned}$$

The theorem follows. □

Theorem 9. *The probability that an agent is idle, i.e. no viewer is now requesting bootstrapping service to it, is*

$$\sum_{b=0}^{m} \frac{(\tau/\gamma)^b}{b![\sum_{i=0}^{m}\frac{1}{i!}(\frac{\tau}{\gamma})^i][\sum_{i=0}^{b}\frac{1}{i!}(\frac{\lambda}{\mu})^i + \sum_{i=b+1}^{m+k}\frac{1}{b^{i-b}b!}(\frac{\lambda}{\mu})^i]}.$$

Proof. Similarly, we can deduce the probability that an agent is idle, denoted as $\Pr(I)$, is the sum of probabilities on the left-most states, i.e.,

$$\begin{aligned}\Pr(I) &= \sum_{b=0}^{m} \Pr(0,b) \\ &= \sum_{b=0}^{m} \frac{(\tau/\gamma)^b}{b![\sum_{i=0}^{m} \frac{1}{i!}(\frac{\tau}{\gamma})^i][\sum_{i=0}^{b} \frac{1}{i!}(\frac{\lambda}{\mu})^i + \sum_{i=b+1}^{m+k} \frac{1}{b^{i-b}b!}(\frac{\lambda}{\mu})^i]}.\end{aligned}$$

The theorem follows. □

Lemma 7. *Let the expected number of viewers that are served by the agent be $\overline{n}_v$, including those that are being served or waiting to be served. We have*

$$\overline{n}_v = \sum_{b=0}^{m} \sum_{a=1}^{m+k} a \Pr(a,b).$$

Lemma 8. *The throughput of the peers that is served by the agent is*

$$\sum_{b=1}^{m} (\sum_{a=1}^{b-1} a\mu \Pr(a,b) + \sum_{a=b}^{m+k} b\mu \Pr(a,b)).$$

Proof. Throughput, denoted as x, is the number of viewers that are served in an unit time, i.e. cumulative service rate based on the service rate on each state. To obtain the expected throughput, we should sum up the multiplication of service rate on each state and the probability of the state, i.e.,

$$x = \sum_{b=1}^{m} (\sum_{a=1}^{b-1} a\mu \Pr(a,b) + \sum_{a=b}^{m+k} b\mu \Pr(a,b)).$$

□

Theorem 10. *The expected waiting time of the viewer, denoted as $\overline{d}_w$, i.e. the time a peer waits to be served by a bootstrapping peer, is*

$$\frac{\sum_{b=0}^{m}\sum_{a=1}^{m+k} a\Pr(a,b)}{\sum_{b=1}^{m}(\sum_{a=1}^{b-1} a\mu\Pr(a,b)+\sum_{b}^{m+k} b\mu\Pr(a,b))}-\frac{1}{\mu}$$

Proof. According to Little's law, the total service delay, is $\overline{n}_v/x$, which is the sum of $1/\mu$, i.e. the expected time a peer spends in service, and $\overline{d}_w$. Combining the results from Lemma 7 and 8, we can carry out

$$\begin{aligned}\overline{d}_w &= \frac{\overline{n}_v}{x}-\frac{1}{\mu}\\ &= \frac{\sum_{b=0}^{m}\sum_{a=1}^{m+k} a\Pr(a,b)}{\sum_{b=1}^{m}(\sum_{a=1}^{b-1} a\mu\Pr(a,b)+\sum_{b}^{m+k} b\mu\Pr(a,b))}-\frac{1}{\mu}.\end{aligned}$$

□

Suppose that a peer will contact j agents for bootstrapping services to avoid agent departure or busy status. To simplify the complexity, we assume there is no bootstrapping agents shared among these agents. The probability that a viewer successfully obtains the service from bootstrapping peers can be expressed by

$$\Pr(S) = (1-(1-\overline{A}_a)^j))(1-\Pr(R)^j).$$

Theorem 11. *The expected churn-induced delay based on our scheme, denoted as $\overline{d}$, can be carried out by*

$$\overline{d} = \Pr(S)(\overline{d}_w+\overline{d}_f+\overline{d}_{pv})+(1-\Pr(S))\overline{d}_b, \tag{4.8}$$

where $\overline{d}_b = \overline{d}_p + \overline{d}_{sc}$ *is the bootstrapping delay by contacting ordinary peers in the channel for connections, including communication delay and delay to get served by ordinary peers,* $\overline{d}_f$ *is the sum of delays between viewer contacting an agent and agent forwarding the service request to bootstrapping peers, and* $\overline{d}_{pv}$ *is the expected transmission delay from bootstrapping peer to the viewer.*

We do not include the time of retrieving ordinary peer list from agents in the channel because this process runs preventively in the background of viewing current channel and makes on delay when peer switches the channel. Additionally, it is worth mentioning recovery delay has a shorter $\overline{d}_{sc}$ than channel-switching delay because the viewer in recovery has already exchanged the chunk information with some other peers still in the network.

4.3 Performance Evaluation

In this section, we evaluate our scheme against previous general scheme by numerical experiments in terms of delay. Then, we check influences of some key parameters to the system by tuning them in the numerical studies.

To evaluate the performance of our scheme against the general scheme, we compare the delay occurring in our scheme, expressed by Equation (4.8), with channel switching delay in the original scheme, i.e., Equation (4.1) and recovery delay in Equation (4.2). Besides the $\overline{d}_A + \overline{d}_{ls}$, we may find the major difference happens between $\overline{d}_w + \overline{d}_f + \overline{d}_{pv}$ and $\overline{d}_b$, where our scheme saves the time by leveraging agents to pre-schedule the downloading plan and utilize the pre-selected bootstrapping peers to provide a quick and fully streaming service in the initial streaming period.

We abbreviate our agent-based scheme as *AS* and the general scheme as *GS*. In the first experiment, we compare AS with GS with the change of viewer arrival rate to the agent. According to [5], we set the typical properties of a P2P streaming system as following: 1)

the expected time a bootstrapping peer will serve a viewer before hand over it to other peers in the network is 6 seconds; 2) the expected lifetime of a bootstrapping peer is 10 minutes; 3) the expected arrival rate of a bootstrapping peer is 1/20; 4) the maximum number of bootstrapping peers that an agent will manage is 100; 5) the number of viewers that will wait for the service from bootstrapping peers, if they're not occupied in serving other viewers, is 10; 6) the expected lifetime of an agent is 10 minutes; 7) the frequency of heartbeat signal to check if agent is still in the network is every 60 seconds; 8) $\overline{d}_A = 0.5$ second; $\overline{d}_{ls} = 0.5$ second; $\overline{d}_p = 1$ second; $\overline{d}_{sc} = 6$ seconds; $\overline{d}_p = 1$ second; $\overline{d}_{pv} = 0.5$ second; and $\overline{d}_f = 2$ seconds. For recovery delay, we change $\overline{d}_{sc}$ to 2 seconds since viewers in recovery has already contacted some other peers for chunk information before interruption occurs.

From Figure 4.4, we can observe that AS significantly outperforms GS in terms of channel switching delay. Especially when the arrival rates are lower than 40, almost half of the delay can be avoided. With the increase of viewer arrival rate, the number of idle bootstrapping peers decreases, which leads to the increase in delay. For recovery delay, AS is better than GS when the arrival rates are lower than 40. When arrival rates goes up, the performance AS almost converges to that of GS. This is because viewer waiting time is longer when bootstrapping peers are busy. We should notice viewer also contacts ordinary peers simultaneously when they contact the agent for streaming service. This strategy make the worst-case performance of AS at least equal to GS. To increase the performance, we can add more agents in the channel so as to keep the viewer arrival rate low.

Now, we compare AS with GS with the change of heartbeat signal frequency. For a better illustration, we use the reciprocal of frequency, i.e. heartbeat signal interval. We keep the setting as the last experiment except that viewer arrival rate is fixed to 25. From Figure 4.5, we can also see AS generally outperforms GS in terms of channel switching delay. As of recovery delay, the difference is very close, about 12%. As we know, heartbeat signal is to check if the agent is still in the network, so a larger interval, i.e. less frequency, will increase the probability that an agent has left the network when a viewer contacts it. In that case, agent will use ordinary peers in the startup process, which saves less time than bootstrapping

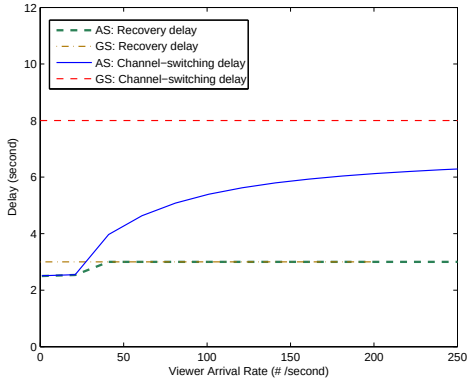

Figure 4.4: Agent-based Scheme v.s. General Scheme when viewer arrival rate changes.

peers. Moreover, it is interesting to see when the heartbeat signal interval is the same as the expected lifetime of an agent, i.e. 10 minutes, AS can save 43% in channel-switching delay.

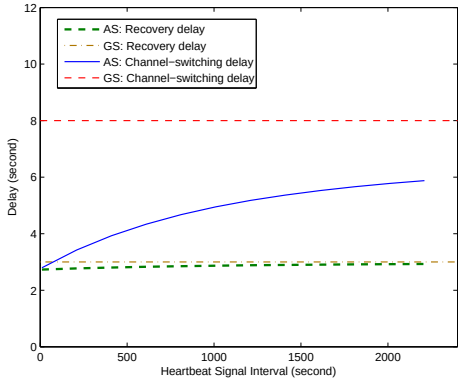

Figure 4.5: Agent-based Scheme v.s. General Scheme when heartbeat signal interval changes.

In the last experiment, we check the influence from the maximum number of bootstrapping peers that can be managed by an agent. As illustrated in Figure 4.6, when the maximum number of bootstrapping peers increases, the delay will decrease. It can be easily understood that more bootstrapping peers will reduce the waiting time when a viewer seeks the service. When the maximum number of bootstrapping peers is lower than 10, the difference between AS and GS in channel-switching delay is only 1.8 seconds at most.

From the above numerical analysis, we can conclude some methods to improve the per-

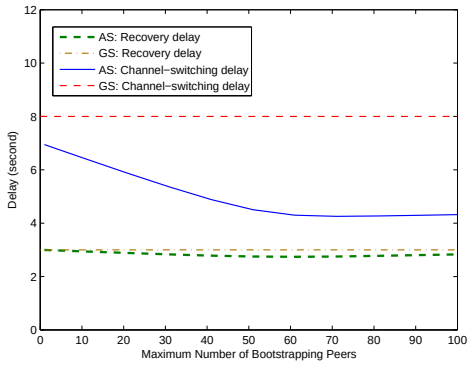

Figure 4.6: Agent-based Scheme v.s. General Scheme when the maximum number of bootstrapping peers changes.

formance of our proposed scheme, which includes adding the number of agents, increasing the heartbeat signal frequency and selecting more bootstrapping peers.

Chapter 5

Summary and Future Work

Our works is motivated by minimizing delays in current P2P streaming system. In this proposal, we focus on two problems:

1) Building delay-minimized overlay streaming mesh. We formulated the minimum-delay P2P streaming problem and presented two solutions for it: a centralized approximation algorithm and a distributed version. We show that our algorithms have a guaranteed performance bound. The distributed version has been extended to adopt to network churn and improve resource utilization. The basic idea of our algorithm can be described by the following. First, we partition the peers V into different clusters according to the regional aggregated streaming capacities. Then, we filter the peers by keeping one representative peer for each cluster, whose streaming capacity is virtualized by the aggregated streaming capacities of the entire cluster. As a result, we form a backbone using the representative nodes. Since the backbone nodes are virtual representatives of clusters, in the next step, we expand the mesh connections from the representative nodes into clusters, which constitutes a final streaming mesh for the overlay network;

2) Devising an agent-based scheme to reduce churn-induced delay. we consider the fact that churn-induced delays identically stem from the time of re-connecting to new peers.

Thus, our scheme propose preventive connections to all channels. Once an actual connection is requested, time will be saved in retrieving bootstrapping information and obtaining authorization as well as authentication. However, maintaining preventive connections for all peers to all channels makes it impractical in terms of the enormous control signals and message overhead. Towards an efficient control and message overhead, we propose that each channel will select some powerful peers as agents to represent the peers in the channel. Agents will distill the bootstrapping peers with superior bandwidth and lifetime expectation to quickly serve the viewer in the initial period of streaming. Moreover, agent will pre-schedule the downloading plan about data chunk partition and streaming for future viewers, and coordinate with each other to build the preventive connections for peers represented by them.

Our contribution can be summarized as following.

(1) To the best of our knowledge, our work represents the first approximation algorithm that optimizes P2P streaming delay and provides an provable upper bound of $O(\sqrt{\log n})$.

(2) We analyze the approximation algorithm's performance and derive the algorithm's approximation ratio, which is found to be the lowest ratio when compared with past results. We not only present an approximation algorithm with a strong theoretical basis, but also reduce the approximation factor to a ratio of $O(\sqrt{\log n})$.

(3) We extend the approximation algorithm to a practical distributed version that is robust to high user churn. Our simulation results indicate our algorithm can actively ensure the end-to-end streaming delay in the worst-case scenario.

(4) We propose the first scheme on reducing the churn-induced delay with analytical model. Our scheme propose preventive connections to all channels. Once an actual connection is requested, time will be saved in retrieving bootstrapping information and obtaining authorization as well as authentication. Our scheme suggests an idea of agent, which facilitates the bootstrapping process in channel switching and peer recovery.

(5) We build a queueing theory model for the agent-based scheme. Based on this model,

we theoretically analyze the performance of our scheme.

(6) We analyze the scheme's performance and derive the scheme's optimal settings based on the numerical experiments. Our numerical experiment results indicate our scheme can significantly reduce the churn-induced delay, especially for the channel-switching delay.

Based on our current direction of research, we propose the following works with the top priority for our post preliminary exam work:

Minimize the end-to-end streaming delay in multi-channel scenario. We propose to devise an algorithm specifically working on multi-channel scenario. Existing heuristics on the problem of reducing P2P streaming delay only focus on the single-channel scenario. However, in most P2P streaming applications, there commonly exists hundreds of channels [7]. As we know, bandwidth allocation and peer assignment in multi-channel streaming pose extra challenge to the minimum-delay problem. Furthermore, current algorithms on the single-channel scenario either provides no theoretical bound on the worst-case performance or loosely estimate the bound without a robust theoretical analysis [13, 14, 19]. The estimated bound of previous algorithms on single channel scenario [19, 20] is $O(\log n)$. Several directions can be considered to reduce the multi-channel streaming delay. For example, by smartly utilizing the bandwidth allocation between channels, we may improve the worst delay in less popular channel while keep the delay in popular channel with a reasonable approximation bound. We propose to design improved clustering-based protocols which considers the optimal bandwidth allocation to achieve a minimum streaming delay. An example design is to decouple the viewing and uploading functionalities on the viewer and reassign the uploading bandwidth in popular channel to improve the streaming delay in less popular channel. By leveraging our current clustering-based approximation, we may guarantee an approximation bound when allocate the bandwidth optimally.

Design a delay-optimized admission control algorithms. Industrial deployment of P2P streaming application may involve different types of peers, including free user, ordinary member paying fees to watch specific channel and VIP member paying extra fees for high-

quality service. However,the traditional scheme in cable TV to prioritize VIP member when bandwidth resource is not sufficient is not applicable. Some free users may have excellent bandwidth to contribute. If we simply put them at the edge of streaming mesh with worst service, we may lost the opportunity to rationally utilize its bandwidth. We propose to devise an admission control algorithm specifically working on multi-type peers. An example solution is to put different reward values when satisfying different types of users and also put rewards based on the streaming delays they have. An optimal solution may well utilize the peers with high bandwidth even they are free or low-level users.

Besides, we are also interesting to solve the following three problems,

Design a large-scale simulation based on our algorithms. P2P networks typically involves hundreds of thousands of users. To evaluate our schemes on large-scale applications, traditional simulation tool like ns-2 is not applicable with memory issues. Some possible solutions emerges as simulation tool exclusively for P2P simulation. For example, PeerSim may handle a simulation for over half-million peers. Such large-scale simulation can evaluates the performance of our distributed scheme with real scalability and may help us identify some improvement directions on the schemes.

Minimize the scheduling delay algorithms. P2P streaming need to exchange data chunk and decide which chunk should transfer with priority. Delayed receiving of chunk may incur delays or interruptions in streaming. So we propose to design a chunk scheduling algorithm to maximize the probability that a chunk will arrive to the viewer before it need to be played. One example solution is that seldom existed chunk should be transferred with priority so more peers may span out this chunk to start a smooth streaming.

Design a gossip-based message exchange scheme. In our current study, gossip is proposed to exchanging agent data and search for resources. We propose to design an improved gossip scheme to ensure a time constraints on message receipts. One possible solution is to utilize super-peers to manage the normal peers and exchange resource information with them. So gossip only needs to run through these super-peers. Since super-peers have long expected

lifetime, i.e. less probability to departure, we can ensure the time constraints in resource searching with certain probability guarantee.

Bibliography

[1] A. Sentinelli, G. Marfia, M. Gerla, L. Kleinrock, and S. Tewari, "Will IPTV ride the peer-to-peer stream?" *Communications Magazine, IEEE*, vol. 45, no. 6, pp. 86–92, June 2007.

[2] R. Kumar, Y. Liu, and K. Ross, "Stochastic fluid theory for p2p streaming systems," May 2007.

[3] S.-Y. Hu, T.-H. Huang, S.-C. Chang, W.-L. Sung, J.-R. Jiang, and B.-Y. Chen, "FLoD: A framework for peer-to-peer 3D streaming," in *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, April 2008.

[4] F. Picconi and L. Massoulie, "Is there a future for mesh-based live video streaming?" in *Peer-to-Peer Computing , 2008. P2P '08. Eighth International Conference on*, Sept. 2008.

[5] X. Hei, C. Liang, J. Liang, Y. Liu, and K. Ross, "A measurement study of a large-scale P2P IPTV system," *Multimedia, IEEE Transactions on*, vol. 9, no. 8, Dec. 2007.

[6] X. Liao, H. Jin, Y. Liu, L. M. Ni, and D. Deng, "Anysee: Peer-to-peer live streaming," in *INFOCOM 2006*, April 2006.

[7] D. Wu, C. Liang, Y. Liu, and K. Ross, "View-upload decoupling: A redesign of multi-channel p2p video systems," in *INFOCOM 2009*, April 2009.

[8] X. Hei, Y. Liu, and K. Ross, "IPTV over P2P streaming networks: the mesh-pull approach," *Communications Magazine, IEEE*, vol. 46, no. 2, pp. 86–92, February 2008.

[9] D. Wu, Y. Liu, and K. Ross, "Queuing network models for multi-channel p2p live streaming systems," in *INFOCOM 2009*, April 2009.

[10] S. Liu, R. Zhang-Shen, W. Jiang, J. Rexford, and M. Chiang, "Performance bounds for peer-assisted live streaming," in *SIGMETRICS '08: Proceedings of the 2008 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, 2008.

[11] V. Venkataraman, P. Francisy, and J. Calandrino, "Chunkyspread: Multitree unstructured peer-to-peer multicast," in *Proceedings of IPTPS06*, 2006.

[12] N. Magharei, R. Rejaie, and Y. Guo, "Mesh or multiple-tree: A comparative study of live P2P streaming approaches," in *INFOCOM 2007*, May 2007.

[13] J. Noh, A. Mavlankar, P. Baccichet, and B. Girod, "Reducing end-to-end transmission delay in P2P streaming systems using multiple trees with moderate outdegree," in *Multimedia and Expo, 2008 IEEE International Conference on*, 23 2008-April 26 2008.

[14] D. Ren, Y.-T. Li, and S.-H. Chan, "On reducing mesh delay for peer-to-peer live streaming," in *INFOCOM 2008*, April 2008.

[15] Z. Chen, K. Xue, and P. Hong, "A study on reducing chunk scheduling delay for mesh-based P2P live streaming," in *Grid and Cooperative Computing, 2008. GCC '08. Seventh International Conference on*, Oct. 2008.

[16] N. Magharei and R. Rejaie, "PRIME: Peer-to-peer receiver-driven mesh-based streaming," in *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, May 2007.

[17] A. da Silva, E. Leonardi, M. Mellia, and M. Meo, "A bandwidth-aware scheduling strategy for P2P-TV systems," in *Peer-to-Peer Computing , 2008. P2P '08. Eighth International Conference on*, Sept. 2008.

[18] A. Caminero, O. Rana, B. Caminero, and C. Carrion, "Improving grid inter-domain scheduling with P2P techniques: A performance evaluation," in *Grid and Cooperative Computing, 2008. GCC '08. Seventh International Conference on*, Oct. 2008.

[19] T. Bonald, L. Massoulié, F. Mathieu, D. Perino, and A. Twigg, "Epidemic live streaming: optimal performance trade-offs," in *SIGMETRICS '08: Proceedings of the 2008 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, 2008.

[20] Y. Liu, "On the minimum delay peer-to-peer video streaming: how realtime can it be?" in *MULTIMEDIA '07: Proceedings of the 15th international conference on Multimedia*. ACM, 2007, pp. 127–136.

[21] F. Huang, B. Ravindran, and V. A. Kumar, "An approximation algorithm for minimum-delay peer-to-peer streaming," in *Peer-to-Peer Computing , 2009. P2P '09. Ninth International Conference on*, Sept. 2009.

[22] C. Wu and B. Li, "rstream: Resilient and optimal peer-to-peer streaming with rateless codes," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 19, no. 1, pp. 77–92, Jan. 2008.

[23] D. Tran, K. Hua, and T. Do, "A peer-to-peer architecture for media streaming," *Selected Areas in Communications, IEEE Journal on*, vol. 22, no. 1, pp. 121–133, Jan. 2004.

[24] G. Dan, V. Fodor, and I. Chatzidrossos, "On the performance of multiple-tree-based peer-to-peer live streaming," in *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, May 2007.

[25] X. Liao, H. Jin, Y. Liu, and L. M. Ni, "Scalable live streaming service based on interoverlay optimization," *Parallel and Distributed Systems, IEEE Transactions on*, Dec. 2007.

[26] E. Brosh, A. Levin, and Y. Shavitt, "Approximation and heuristic algorithms for minimum-delay application-layer multicast trees," *IEEE/ACM Transaction on Networking*, vol. 15, no. 2, pp. 473–484, 2007.

[27] S. Tayu, T. G. Al-Mutairi, and S. Ueno, "Cost-constrained minimum-delay multicasting," in *SOFSEM 2005: Theory and Practice of Computer Science*. Springer, Berlin Heidelberg, 2005, pp. 330–339.

[28] J. Könemann, A. Levin, and A. Sinha, "Approximating the degree-bounded minimum diameter spanning tree problem," *Algorithmica*, vol. 41, no. 2, pp. 117–129, February 2005.

[29] B. Brinkman and M. Helmick, "Degree-constrained minimum latency trees are APX-Hard," Miami University, Tech. Rep., 2008.

[30] G. Pandurangan, P. Raghavan, and E. Upfal, "Building low-diameter peer-to-peer networks," *Selected Areas in Communications, IEEE Journal on*, vol. 21, no. 6, pp. 995–1002, Aug. 2003.

[31] Y. Kim, J. K. Park, H. J. Choi, S. Lee, H. Park, J. Kim, Z. Lee, and K. Ko, "Reducing IPTV channel zapping time based on viewer surfing behavior and preference," in *Broadband Multimedia Systems and Broadcasting, 2008 IEEE International Symposium on*, 31 2008-April 2 2008.

[32] C. Cho, I. Han, Y. Jun, and H. Lee, "Improvement of channel zapping time in iptv services using the adjacent groups join-leave method," in *Advanced Communication Technology, 2004. The 6th International Conference on*, 2004.

[33] H. Joo, H. Song, D.-B. Lee, and I. Lee, "An effective iptv channel control algorithm considering channel zapping time and network utilization," *Broadcasting, IEEE Transactions on*, vol. 54, no. 2, pp. 208–216, June 2008.

[34] H. Fuchs and N. Farber, "Optimizing channel change time in iptv applications," in *Broadband Multimedia Systems and Broadcasting, 2008 IEEE International Symposium on*, 31 2008-April 2 2008.

[35] Y. He, G. Shen, Y. Xiong, and L. Guan, "Optimal prefetching scheme in p2p vod applications with guided seeks," *Multimedia, IEEE Transactions on*, vol. 11, no. 1, pp. 138–151, Jan. 2009.

[36] B. Cheng, X. Liu, Z. Zhang, and H. Jin, "A measurement study of a peer-to-peer video-on-demand system," in *IPTPS*, 2007.

[37] V. Fodor and G. Dan, "Resilience in live peer-to-peer streaming [peer-to-peer multimedia streaming]," *Communications Magazine, IEEE*, vol. 45, no. 6, pp. 116–123, June 2007.

[38] I. Hernandez-Serrano, S. Sharma, and A. Leon-Garcia, "Reliable p2p networks: Treblecast and treblecast," in *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, May 2009.

[39] D. Qiu and W. Sang, "Global stability of peer-to-peer file sharing systems," *Computer Communications*, vol. 31, no. 2, pp. 212 – 219, 2008, special Issue: Foundation of Peer-to-Peer Computing.

[40] Y. Tian, D. Wu, and K. W. Ng, "Modeling, analysis and improvement for bittorrent-like file sharing networks," in *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, April 2006.

[41] M. Hefeeda and O. Saleh, "Traffic modeling and proportional partial caching for peer-to-peer systems," *Networking, IEEE/ACM Transactions on*, vol. 16, no. 6, pp. 1447–1460, Dec. 2008.

[42] E. M. Arkin, M. A. Bender, S. P. Fekete, J. S. B. Mitchell, and M. Skutella, "The freeze-tag problem: How to wake up a swarm of robots," *Algorithmica*, vol. 46, no. 2, pp. 193–221, October 2006.

[43] M. Sztainberg, E. Arkin, M. Bender, and J. Mitchell, "Theoretical and experimental analysis of heuristics for the "freeze-tag" robot awakening problem," *Robotics, IEEE Transactions on*, vol. 20, no. 4, pp. 691–701, Aug. 2004.

[44] F. Lam and A. Newman, "Traveling salesman path problems," *Mathematical Programming*, vol. 113, no. 1, pp. 39–59, May 2008.

[45] C. H. Papadimitriou and S. Vempala, "On the approximability of the traveling salesman problem," *Combinatorica*, vol. 26, no. 1, pp. 101–120, Feb. 2006.

[46] J. Peltotalo, J. Harju, A. Jantunen, M. Saukko, L. Vaatamoinen, I. Curcio, I. Bouazizi, and M. Hannuksela, "Peer-to-peer streaming technology survey," in *Networking, 2008. ICN 2008. Seventh International Conference on*, April 2008.

[47] X. Liu, H. Yin, C. Lin, and C. Du, "Efficient user authentication and key management for peer-to-peer live streaming systems," *Tsinghua Science & Technology*, vol. 14, no. 2, pp. 234 – 241, 2009.

[48] B. Li, S. Xie, Y. Qu, G. Keung, C. Lin, J. Liu, and X. Zhang, "Inside the new coolstreaming: Principles, measurements and performance implications," in *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, April 2008.

[49] C. Wu, B. Li, and S. Zhao, "Multi-channel live p2p streaming: Refocusing on servers," in *INFOCOM 2008*, April 2008.