On Reducing Delays in P2P Live Streaming Systems

Fei Huang

Dissertation 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

September 3, 2010
Blacksburg, Virginia

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

On Reducing Delays in P2P Live Streaming Systems

Fei Huang

(ABSTRACT)

In the recent decade, peer-to-peer (P2P) technology has greatly enhanced the scalability of multimedia streaming on the Internet by enabling efficient cooperation among end-users. However, existing streaming applications are plagued by the problems of long playback latency and long churn-induced delays. First of all, many streaming applications, such as IPTV and video conferencing, have rigorous constraints on end-to-end delays. Moreover, churn-induced delays, including delays from channel switching and streaming recovery, in current P2P streaming applications are typically in the scale of 10-60 seconds, which is far below the experience users expect in applications such as cable TV systems. These two issues of playback latency and churn-induced delays have hindered the extensive commercial deployment of P2P systems. Motivated by this, in this dissertation, we focus on reducing delays in P2P live streaming systems. Specifically, we propose solutions for reducing delays in P2P live streaming systems in four problem spaces: (1) minimizing the maximum end-to-end delay in P2P streaming; (2) minimizing the average end-to-end delay in P2P streaming; (3) minimizing the average delay in multi-channel P2P streaming; and (4) reducing churn-induced delays.

We devise a streaming scheme to minimize the maximum end-to-end streaming delay under a mesh-based overlay network paradigm. We call this problem the MDPS problem. We formulate the MDPS problem and prove its NP-completeness. We then present a polynomial-time approximation algorithm, called Fastream-I, for this problem, and show that the performance of Fastream-I is bounded by a ratio of $O(\sqrt{\log n})$, where n is the number of peers in the system. We also develop a distributed version of Fastream-I that can adapt to network dynamics. Our simulation study reveals the effectiveness of Fastream-I, and shows a reasonable message overhead.

While Fastream-I yields the minimum maximum end-to-end streaming delay (within a factor of $O(\sqrt{\log n})$), in many P2P settings, users may desire the minimum average end-to-end P2P streaming delay. Towards this, we devise a streaming scheme which optimizes the bandwidth allocation to achieve the minimum average end-to-end P2P streaming delay. We call this problem the MADPS problem. We first develop a generic analytical framework for the MADPS problem. We then present *Fastream-II* as a solution to the MADPS problem. The core part of Fastream-II is a fast approximation algorithm, called *APX-Fastream-II*, based on primal-dual schema. We prove that the performance of APX-Fastream-II is bounded by a ratio of $1 + \omega$, where ω is an adjustable input parameter. Furthermore, we show that the flexibility of ω provides a trade-off between the approximation factor and the running time of Fastream-II.

The third problem space of the dissertation is minimizing the average delay in multi-channel P2P streaming systems. Toward this, we present an algorithm called Fastream-III. To re-

duce the influence from frequent channel-switching behavior, we build Fastream-III for the view-upload decoupling (VUD) model, where the uploaded content from a serving node is independent of the channel it views. We devise an approximation algorithm based on primal-dual schema for the critical component of Fastream-III, called APX-Fastream-III. In contrast to APX-Fastream-II, APX-Fastream-III addresses the extra complexity in the multichannel scenario and maintains the approximation bound by a ratio of $1 + \omega$.

Besides playback lag, delays occurring in P2P streaming may arise from two other factors: node churn and channel switching. Since both stem from the re-connecting request in churn, we call them churn-induced delays. Optimizing churn-induced delays is the dissertation's fourth problem space. Toward this, we propose NAP, a novel agent-based P2P scheme, that provides preventive connections to all channels. Each channel in NAP selects powerful peers as agents to represent the peers in the channel to minimize control and message overheads. Agents 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 NAP. Based on this model, we numerically compare NAP's performance with past efforts. The results of the numerical analysis reveal the effectiveness of NAP.

This work has been partially supported by NSF Nets Grant CNS- 0626964, NSF HSD Grant SES-0729441, NIH MIDAS project 2U01GM070694-7, NSF PetaApps Grant OCI-0904844, DTRA R&D Grant HDTRA1-0901-0017, DTRA CNIMS Grant HDTRA1-07-C-0113, NSF NETS CNS-0831633, DHS 4112-31805 and DOE DE-SC0003957.

Dedication

I dedicate this dissertation to my family. Without their love and support, this work would not have been possible.

In particular, I dedicate this work to the loving memory of my grandpa – HUANG, Shan (黄山), grandma – BAO, Huying (鲍沪英), and my uncle – ZHU, Shu (朱澍).

此论文献给我挚爱的家人。感谢你们的爱与支持。

特别缅怀我的爷爷(黄山)，奶奶(鲍沪英)和舅舅(朱澍)。

Acknowledgments

After several years of efforts, I finally reach the moment to accomplish this dissertation. I would like to take this opportunity to acknowledge my mentors, friends, and family for their helps and supports all the way in my study.

First, many thanks should be presented to my advisor, Dr. Binoy Ravindran, who not only opens the door for my PhD research in the optimization algorithms for P2P streaming systems, but also provides all his supports, academically and personally, to the success of this dissertation. Strong acknowledgement should also be delivered to Dr. Anil Vullikanti and Dr. Maleq Khan, who inspired me with many worthy suggestions in the accomplishment of this work.

Also, I am grateful to Dr. Achla Marathe and Dr. Madhav V. Marathe. They brought me to the Network Dynamics and Simulation Science Laboratory, where I have the opportunity to study extensive research ideas and explore multiple projects in network science. Such research experience makes an important constituent in my entire PhD studies. I would also acknowledge Dr. Thomas Hou, Dr. Douglas Jensen, Dr. Subhash Sarin and Dr. Yaling Yang for serving in my committee. They provided big helps in the completion of this dissertation. In addition, I would thank my colleagues and friends at Virginia Tech. Their friendship and frank suggestions encouraged me all the way.

Last but not least, I would express my acknowledgement to my family, who support me all the way with their loves. My father and mother, Mingshu Huang (黄明树) and Chengyun Zhu (朱承云), always encourage my confidence to pursue my academic interest with their entire loves. My grandparents, En'yuan Zhu (朱恩源) and Meijuan Sun (孙美娟), taught me to read words since I was very young. They enlightened me on the essential skills to explore this wonderful world. My wife, Min Hu (胡敏), devote her love and considerate support to me all the way. No words can elaborate my appreciation to my family. This work is to them.

Contents

List of Figures

List of Tables

Chapter 1

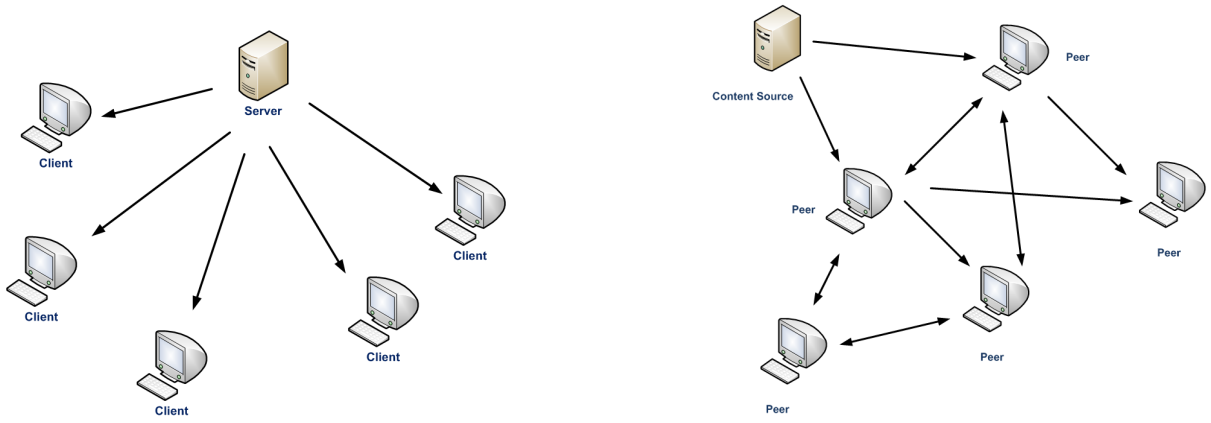
Introduction

In the recent decade, peer-to-peer (P2P) networks have greatly enhanced the multimedia content distribution on the Internet by enabling efficient cooperation among end users [1, 2]. Compared with the traditional client-server model, where only servers supply the content , and clients passively receive the content, peers in P2P model play a double role as suppliers and receivers at the same time. From Figure 1.1, we can see peers not only receive the data but also contribute their bandwidth and storage resources to help forward the data to other peers in the network. In contrast to the constrained resources of the client-server model, peer cooperations greatly improve resource availability, and thus massively increase network scalability.

1.1 Problems and Motivation

Benefiting from the significant scalability, there is an increasing demand for applications of P2P live streaming, such as IPTV, VOIP, and video conferencing [3–7]. As a result, many P2P live video systems, such as PPLive, PPS, and Sopcast, etc., have been successfully deployed in the recent years, and most of them have over 100 channels, with millions of users [8–12]. 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 [13–16]. Motivated by this, in this dissertation, we work towards reducing four types of streaming delays in P2P networks, including (1) minimizing the maximum end-to-end delay in P2P streaming, (2) minimizing average end-to-end delay in P2P streaming, (3) minimizing average delay in multi-channel P2P streaming, and (4) reducing churn-induced delays.

P2P protocols are applicable in a wide variety of application contexts, the most familiar including streaming media (such as IPTV and video conferencing), file sharing, and cloud computing. Less familiar are many applications in military contexts - for example, the dis-



(a) (b)

Figure 1.1: Streaming infrastructures: (a) Client-Server model; (b) Peer-to-Peer model.

semination of time-critical sensor (e.g., radar ground moving target information (GMTI) and synthetic aperture radar (SAR) imagery) from an airborne sensor platform to surveillance consumers [17]. Unlike the better known and more widely used P2P applications, many military applications have P2P latency constraints that are safety-critical, regardless of whether the time frames are seconds, minutes, or hours. Very little publicly available research addresses timeliness of P2P protocols.

Current works on P2P streaming can be broadly classified into two classes: (1) tree-based overlays, and (2) mesh-based overlays [9, 18–22]. Recent studies have shown that the mesh-based approach consistently exhibits a superior performance over the tree-based approach [19, 23]. The tree-based P2P streaming approach generally organizes peers into multiple diverse trees. After obtaining the description of a Multiple Description Coded (MDC) content, it pushes each description through separate trees [18, 19]. In contrast, the mesh-based P2P streaming approach arranges peers into a randomly connected mesh and employs swarming content delivery [19]. The major advantages of mesh-based systems are easy maintenance and inherent robustness in high-churn P2P environment [9]. Motivated by these promising advantages, we study the minimum-delay problems under the mesh-based model. Although our algorithms are developed with a mesh paradigm, they are ready to fit the multiple tree-based model after simple modifications as well.

First, we focus on minimizing the maximum playback lag, i.e. maximum 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 to meet such delay constraints in highly dynamic and heterogenous P2P network environments is an open problem that poses multiple challenges. 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 [1,24,25].

Motivated by this, our first target is to minimize the maximum end-to-end streaming delay in P2P networks. To arrive at a tractable solution, our work focuses only on minimizing the communication delay.

Minimizing the maximum streaming delays in P2P networks is an NP-complete problem. This is due to heterogeneous bandwidth requirements and network dynamics of P2P systems [26–28]. Thus, obtaining optimal solutions to this problem for large-scale networks is intractable. In this dissertation, we propose an efficient approximation algorithm, called *Fastream-I*, for this problem. Fastream-I *provably* achieves a delay assurance by an approximation ratio of $O(\sqrt{\log n})$, where n represents the number of peers in the network [29]. 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.

Second, we tackle the problem of minimum average delay in P2P streaming. Recently, layered coding has emerged as a viable solution for delivering real-time streaming content [30, 31]. This technique not only provides an adaptive support for different downloading capacities of peers, but also allows IPTV service providers to deliver live content at diverse video definitions from the same coding process. For example, viewers may pay general fees for a standard service, or extra fees for 1080HD video or even 3D video. Unlike traditional IPTV service where viewers only download the multimedia content [32], under the P2P paradigm, substantial bandwidth may be available to viewers who pay only for a standard service, while HD viewers may instead suffer from bad streaming service due to the bandwidth deficit among them. To maximize the bandwidth utilization, we should enable peer cooperation among viewers of different service qualities. To deal with that, the HD content can be forwarded through peers with standard service, but only the HD viewers receive the authorization key for viewing HD content. This raises a fundamental question: how to optimally distribute the video content and conduct sub-stream scheduling among peers with diverse service qualities, while achieving the minimum average end-to-end P2P streaming (or MADPS) delay [33, 34]. We call this problem the MADPS problem.

To address this problem, we devise a streaming scheme, called *Fastream-II*, which is capable of optimizing the bandwidth allocation to achieve the minimum average end-to-end P2P streaming delay. Minimizing average streaming delay for P2P live systems is not a trivial problem. Previous theoretical works on designing P2P live streaming usually assume a homogeneous service quality [21, 29]. Thus, obtaining optimal solutions to this problem for large-scale networks is expensive in terms of algorithmic computational costs [35]. Approximate or heuristic solutions with scalable costs are therefore highly desirable. In this dissertation, we focus on approximate algorithms because we target time-critical P2P applications (e.g., video conferencing, or cloud computing), for which assured bounds on performance and running time are more desirable than heuristic (or empirically-established). In addition, the analytical foundation that is necessary for developing approximate algorithms can contribute to a greater understanding of the problem and can provide deeper insights on designing efficient algorithms, be they approximate or heuristic. We take the first such steps toward this [36].

For a feasible solution, we start with the assumption of a static network—i.e., no churn. In this way, we can devise a framework which is analytically achievable. The method will be most suitable for the scenario where a service provider deploys a set-top box at viewers' homes. In that case, even when a viewer turns off the TV, the set-top box can still contribute its bandwidth to other viewers. For this scenario, we first develop an analytical model that formulates the MADPS problem as an optimization problem. Then we propose Fastream-II to solve MADPS problem. Inspired by the primal-dual schema, we develop an approximation algorithm as the core of Fastream-II, called *APX-Fastream-II* for optimally utilizing the bandwidth among peers subscribing to different video qualities, while achieving the minimum average streaming delay. We show that APX-Fastream-II's performance is bounded by a factor of $1+\omega$, where ω is an input parameter. Fastream-II's running time is also bounded. We show that there exists a trade-off between APX-Fastream-II's approximation factor ω and its running time. The approximation factor is adjustable in the range of $(1, n]$, where n is the number of peers in the network. This trade-off allows users to flexibly tune the performance bound according to running time requirements.

Third, we devise a solution to minimizing average delay in multi-channel P2P streaming. Traditionally, multi-channel streaming systems organize peers into multiple overlays, where each overlay is formed by a swarm of peers viewing the same channel [8, 22, 37]. Peers in the same overlay exchange data exclusively inside the swarm. We call such a scheme of isolated multiple overlays as IMO. Once a viewer j switches from channel A to channel B, it needs to join the overlay of channel B and find new data feed from the swarm B. At the same time, peer j may stop its service in channel A. As a result, peers receiving data from j in the swarm of A have to reconnect to other peers in swarm A. Recent studies [5, 8, 38, 39] indicate fundamental performance problems for IMO-based systems. First, the frequent channel switching (or channel churn) undermines the viewing reliability under the IMO model. Furthermore, the upload bandwidth resources of different channels are highly unbalanced, where channels with low bandwidth suffer poor performance. (The IMO model does not enable bandwidth sharing from channels with surplus bandwidth.) Thus, minimizing multi-channel delays based on the IMO model cannot provide an effective solution. In other words, once any viewer changes the channel, the minimum-delay streaming topology need to change accordingly.

Recently, Wu *et al.* proposed a *view-upload decoupling* (or VUD) model to improve the streaming qualities on all channels [8]. The VUD-based P2P streaming approach decouples the role of peers into viewing and uploading. In other words, what a peer uploads is independent of what it views and subscribes to [13]. The major advantages of VUD-based systems are inherent robustness in the high-channel-churn environments and flexible cross-channel resource sharing capability. In light of these promising advantages, we study the minimum-delay problem under the VUD-based model. The problem of how to share the bandwidth across different channels toward minimizing the streaming delay is an open problem.

We call the set of peers uploading data for channel A the *serving swarm* A. As a common strategy in existing P2P protocols [40, 41], when a viewer j connects to channel A, it will

contact the server or its neighbors for a list of peers in the serving swarm A. We call such a set of peers *ports* because they function as a virtual port to feed the data to others. Besides ports, there is another category of peers inside the serving swarm. They forward the content directly or indirectly to the ports. We call them *internal servants*. For simplicity, we assume that internal servants only upload data to peers inside the serving swarm, while ports only feed data to viewers. We do not provide the methodology on how to select the ports from the serving swarm. Generally, we can employ existing solutions in terms of bootstrapping peers or super peers as in [40, 42].

Since the viewers may switch channels dynamically, we focus on minimizing the streaming delay from the sources to the ports. Our strategy is to obtain the minimum source-to-port delay streaming (MSPDS) by devising an optimal streaming topology and bandwidth allocation algorithm among different serving swarms. We call this problem the MSPDS problem. For the MSPDS problem, we propose an algorithm, called Fastream-III, to minimize the average delay in multi-channel P2P streaming systems. To reduce the influence from frequent channel-switching behavior, we build Fastream-III for the view-upload decoupling (VUD) model, where the uploaded content from a serving node is independent of the channel it views. We devise an approximation algorithm based on primal-dual schema for the critical component of Fastream-III, called APX-Fastream-III. In contrast to APX-Fastream-II, APX-Fastream-III tackles the extra complexity in multichannel scenario and maintains the approximation bound by a ratio of $1 + \omega$.

Having developed algorithms for the no-churn case, we turn our attention to P2P applications with high network churns. Thus, in the last part of this work, we propose *NAP*, a Novel, Agent-based P2P scheme to reduce the churn-induced delay. As we know, besides playback lag, delays occurring in P2P streaming may arise from two other factors: node churn and channel switching [13,43]. 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 [13]. 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 [8], 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 swarm 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 are accustomed to delays under seconds, which are typical in a cable TV system [8]. However, churn-induced delays are significant in current P2P systems. For example, measurement studies indicate that channel switching delays are typically on the scale of 10-60 seconds [9]. From the perspective of user experience, this is obviously undesir-

able. Motivated by this, we propose a novel agent-based P2P architecture to preventively reduce the churn-induced delay [40, 44]. The scheme built from this dissertation can adopt our minimum-playback-delay schemes: Fastream series [29, 36, 45] or any of the existing algorithms, such as [21, 35], to yield low-delay streaming.

Based on the fact that churn-induced delays identically stem from the time cost of re-connecting to new peers, NAP is devised with preventive connections to all channels. Once an actual connection is requested, time is saved in retrieving bootstrapping information, and obtaining authorization and authentication. However, maintaining preventive connections for all peers to all channels makes it impractical due to significant number of control signals and message overhead. To achieve an efficient control and reduced message overhead, we propose that for each channel some powerful peers are selected as agents to represent the peers in the channel. The agents distill the bootstrapping peers with superior bandwidth and lifetime expectation to quickly serve the viewer in the initial period of streaming. Moreover, the agents 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. Since agents work in a distributed manner and their amount fluctuates adaptively to the network size, NAP exhibits massive scalability. In this dissertation, we will not discuss security concerns. NAP can adopt existing security schemes, which have been extensively studied in previous works [46, 47].

1.2 Summary of Contributions

In summary, our main goal in this research is to design schemes to reduce the major delays that occur in P2P streaming. Our work makes important contributions in four problem spaces:

(1) Fastream-I for minimizing maximum end-to-end delay: to the best of our knowledge, our work represents the first approximation algorithm that optimizes the maximum end-to-end P2P streaming delay [29]. Past work regarding minimum delay P2P streaming mainly builds on heuristics, which either provide no theoretical bound on the worst-case performance or loosely estimate the bound without a robust theoretical analysis. In contrast, Fastream-I provides a 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. 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 $O(\sqrt{\log n})$. 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.

(2) Fastream-II for minimizing average end-to-end delay: we present a fast approximation-based algorithm for the MADPS problem, called Fastream-II [36, 45]. To the best of our knowledge, we are not aware of any other approximation solution to the MADPS problem; ours is the first . There is no known efficient algorithm with a practically-feasible running time to solve this problem optimally. In contrast, Fastream-II guarantees a polynomial-time bound on the running time. We show that the critical component – APX-Fastream-II of Fastream-II – has performance that is bounded by a factor of $1 + \omega$, where ω is an input parameter. This bound is found to be the lowest ratio when compared with past results. We show that a trade-off exists between the algorithm's approximation factor ω and its running time, which allows users to flexibly tune the appropriate performance bound according to running time requirements. This proposed algorithm also works for heterogeneous scenarios, where viewers pay for different streaming qualities, which is usually overlooked in previous works as they assume a homogeneous quality. Our algorithm optimally utilizes the bandwidth resources from all peers in the network, not just those who pay for the service. We implement simulations based on Fastream-II. The results indicate our proposed approach can actively ensure the end-to-end streaming delay in the worst-case scenario.

(3) Fastream-III for minimizing average multi-channel delay: we present a polynomial-time algorithm – Fastream-III – for the MSPDS problem. We develop a tractable analytical framework for the MSPDS problem, which can provide an analytical foundation for algorithm design. We devise the critical part of Fastream-III by a fast approximation algorithm based on primal-dual schema, called APX-Fastream-III. One of the steps in APX-Fastream-III involves shortest path search. We extend the shortest path search from each step for just searching one node in Fastream-II to each step for building a shortest path tree to all nodes. This improvement greatly facilitates the algorithm efficiency. Finally, we show that the performance of APX-Fastream-III is also bounded by a factor of $1 + \omega$, where ω is an input parameter. We show that there exists a trade-off between the APX-Fastream-III's approximation factor ω and its running time, which we can tune the appropriate performance bound according the requirement on running time. To the best of our knowledge, we are not aware of any other solution to the MSPDS problem.

(4) NAP for reducing churn-induced delays: our work to reduce the churn-induced delay provides the first scheme on this topic with an analytical model and numerical evaluation for the performance [40]. 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 an agent, which facilitates the bootstrapping process in channel switching and peer recovery. We build a queueing theory model for the agent-based scheme, which provides theoretical

foundation for NAP and for problems in similar category. Based on this model, we theoretically analyze the performance of our scheme. 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 Outline

The rest of this dissertation is organized as follows. Chapter 2 describes an overview of past and related works. Chapter 3 introduces the preliminary knowledge and general network models that are used in this dissertation. In Chapter 4, we describe and formulate the MDPS problem. We prove its NP-completeness and then present our proposed approximation algorithm Fastream-I and derive its performance guarantee. We compare Fastream-I against past works through simulation-based experimental studies. Chapter 5 presents our network model and formulates the MADPS problem. Then, we devise our proposed solution Fastream-II based on the primal-dual schema in approximation algorithm. Furthermore, we derive the approximation performance and running time of APX-Fastream-II, the core component of Fastream-II. In Chapter 6, we describe and formulate the MADPS problem. We devise Fastream-III by way of primal-dual schema. Then, we prove the approximation bound of APX-Fastream-III, the core component of Fastream-III. Chapter 7 describes the problem of reducing churn-induced delay, i.e. channel-switching delay and recovery delay, and propose our agent-based scheme. We model NAP by a queuing theory model. Based on that, we numerically study the performance our scheme against the general scheme. Chapter 8 summarizes the work in this dissertation and describes our post-preliminary-exam work.

Chapter 2

Related Work

In this section, we look at the past work related to our problem space. First, we describe the general background on the mainstream streaming topologies. Then, we review the related work in terms of minimum streaming delay and churn-induced delays.

2.1 Background

P2P networks build at the application layer, on top of the underlying physical network topology. Such design makes the P2P system independent of the physical network topology [48,49]. Thus, we normally call a P2P network an overlay network as well. Existing P2P streaming strategies can be broadly classified into two categories: (1) *tree-based overlays*, and (2) *mesh-based overlays* [9,18–22].

2.1.1 Tree-based Overlays

In the early stage, the tree-based overlays are devised as a *single-tree model*, where peers are organized into a tree structure rooted at the source node [50]. As illustrated in Figure 2.1a, each peer forwards the data it receives from the parent node to its child nodes. Since every peer only has one parent node, such design imposes a strict requirement on the bandwidth of the parent node, which should be at least larger than the streaming rate. Another major drawback of this model is that the leaf nodes do not contribute their bandwidth resources to the tree . From graph theory, we understand the number of leaf nodes accounts for at least 50% of the total nodes in the network, given the out-degree on node is larger than 2. Therefore, the single-tree model degrades the bandwidth utilization efficiency [51].

For better bandwidth utilization, the tree-based P2P streaming approach is generally used to organize peers into multiple diverse trees [18,52]. We call such streaming topology models

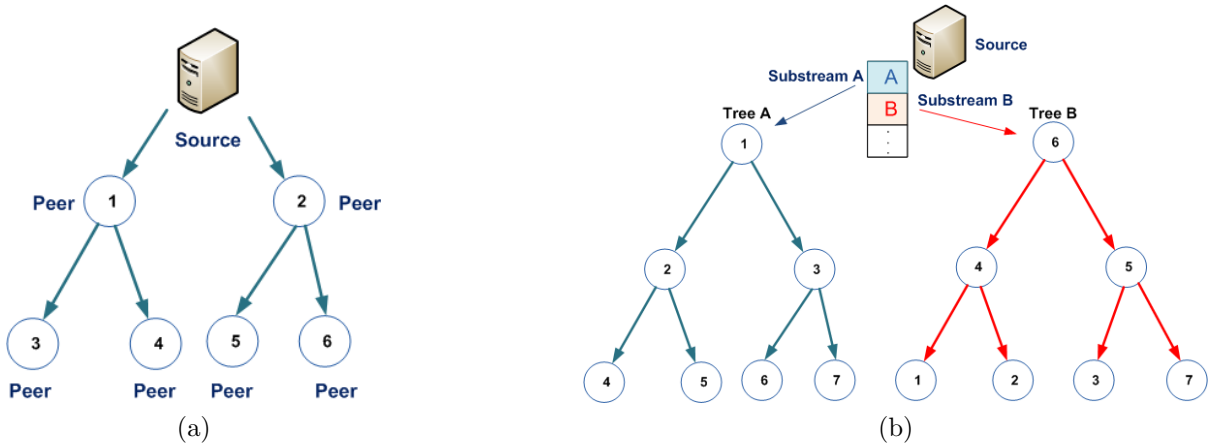


Figure 2.1: Tree-based overlays: (a) Sing-Tree model; (b) Multi-Tree model.

the *multi-tree model*. As shown in Figure 2.1b, in the multi-tree model, the source node divides the stream into multiple substreams. Instead of one single tree, multiple trees are built, where each tree is formed to distribute one substream respectively. Each peer joins all the subtrees to retrieve substreams. To reduce the influence from node churn and effectively utilize available bandwidth resources, each peer is positioned as an internal node in only one tree, and as a leaf node in other trees. Then, content of each substream is disseminated through a specific tree. Such content delivery method is also called a *push mechanism*, where internal peers in each tree simply forward received data to the child nodes [19]. The multi-tree model is well organized from the perspective of a streaming topology. However, the management of streaming trees is vulnerable to frequent node churns. Once a node joins or departs the network, the trees where they reside need maintenance update .

2.1.2 Mesh-based Overlays

The mesh-based P2P streaming approach arranges peers into a randomly connected mesh and employs swarming content delivery [19,53]. Unlike the tree-based overlay, the streaming topology in mesh-based overlay is not static. As shown in Figure 2.2, peers in the network can download/upload data content from/to multiple nodes simultaneously [51]. Such a content delivery method is also called a *pull mechanism*, where new arrival nodes initiate the request of streaming from other peers. The major advantages of mesh-based systems are easy maintenance, high bandwidth utilization and inherent robustness in high-churn P2P environment [9]. However, previous studies, e.g., [51], found the delay performance of current mesh-based strategies is not satisfactory. This motivates us to study minimum delay problem under this model.

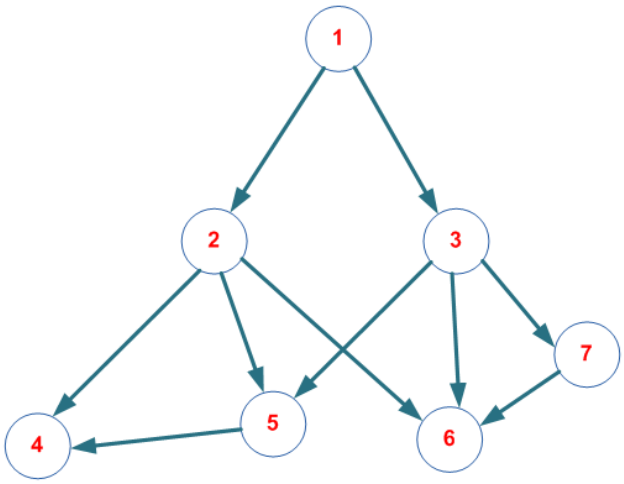


Figure 2.2: Mesh-based overlays.

2.2 Strategies on Minimum-Delay P2P Streaming

Theoretical works on the minimum delay P2P streaming problem are limited, though recently a growing number of studies have focused on P2P live streaming [21, 22, 29, 35, 54, 55]. Most of the previous protocols for the P2P streaming delay problem are based on tree-shaped overlays [6, 18, 20, 52, 56]. Tran *et al.* proposed Zigzag in [56], 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 [20], Noh *et al.* proposed an overlay consisting of multiple trees with moderate out-degree to reduce end-to-end transmission delays in the P2P media streaming systems. In [18], Venkataraman *et al.* presented 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 [19]. In addition, it is difficult to obtain the globally optimal delay and coordinate among different trees [21]. 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 [57].

Recently, an increasing number of studies have focused on mesh-based P2P live streaming [21, 22, 35, 55, 58]. For example, in [35], Wu *et al.* [35] presented 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.

Due to the lack of formal theoretical bounds, intuitions and heuristics have driven the design of P2P schemes so far [21, 54]. However, 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. For example, Ren *et al.* [21] proposed a heuristic to reduce the delay on mesh topology, where peers select their parents based on the metric of link capacity divided by communication delay. In this algorithm, peers located at the edge of mesh may only download the data without uploading, which may lead to low bandwidth utilization in P2P networks. Thus, when the total uploading capacity is close to the downloading capacity in the P2P community, some peers may not be able to receive live streaming. The heuristic does not provide performance guarantees on the end-to-end streaming delay, which is important in time-critical applications, such as video conferencing.

Furthermore, most of the current algorithms on the P2P streaming scenario [20,21,59] either provide no theoretical bound on the worst-case performance or loosely estimate the bound without a robust theoretical analysis. The estimated bound of previous algorithms [55, 59] is $O(\log n)$. In contrast, we not only present an approximation algorithm, called Fastream-I, with a strong theoretical basis, but also provide a near-optimal approximation factor of $O(\sqrt{\log n})$.

In [51, 55], Liu *et al.* proposed a chunk-based P2P video streaming strategy, called *snowball* streaming algorithm to achieve low streaming delay. They look on P2P streaming as a process of continuous chunk generation. Based on that, they focus on the minimum delay required to disseminate each chunk in streaming. The snowball algorithm has strong assumption that on each time unit during streaming only one chunk is generated and the upload bandwidths of peers are the integral multiples of chunk size. However, in practical P2P scenario, multiple chunks may be generated inside one time unit and the bandwidths of peers may not follow integral units given the chunk size as the unit bandwidth. Although we can round the bandwidth to integral units, significant bandwidth from the fractional part will be ignored. Cumulatively, large amount of bandwidth resource will be wasted. In addition, the snowball algorithm only provides a lower bound on the delay, which represents the best performance of the snowball algorithm and can be used as a benchmark for other algorithms. However, time-critical applications normally put more attention on the strategies that can guarantee the worst-case performance. Furthermore, snowball algorithm minimizes the delay from the perspective of chunk dissemination time. Although it considers the impact of communication delays to the algorithm, it does not minimize such delays. In contrast, our solutions provide the assurance on the worst cast performance and takes the communication delay as the major fact in measurement.

In [60], Ezonvski *et al.* proposed a strategy to minimize the average time required to deliver a file to the receivers. This method employs the water-filling technique to sequentially minimize the finish times on individual peers, given constrained upload capacities of peers [61]. They prove such sequential minimization procedure as an intermediate step can bring an optimal schedule strategy for the file transmission to all peers. However, the problem of minimizing average finish time in file transmission is significantly different from the minimum-delay

P2P streaming problem. File transmission only requires the complete collection of all data segments without considering the sequence they are received. In a P2P live streaming session, a sequence of video segments are continuously generated by the source node and disseminated to all peers. For a continuous playback, data segments should be received in the original time sequence at the viewers. Therefore, it is apparent that the method in [60] can only be utilized when the video content can be completely downloaded before viewing. For live streaming, this method is not appropriate.

The problem of minimizing maximum end-to-end delay in P2P streaming is called MDPS problem. It has some similarity with the minimum-delay multicast tree problem (MDMT problem) [62, 63] and degree-bounded minimum-diameter tree problem (DBMDT problem) [64–66]. Fastream-I is inspired by the clustering method first proposed by Könemann *et al.* [64]. 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 Fastream-I will generate a subgraph which is not simply a tree. Besides, the proofs on DBMDT theorems in [64] 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 [64] 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.

The minimum average delay P2P streaming problem, or called MADPS problem has some similarity with the minimum-cost multi-commodity flow problem (or MCMF) [67,68]. Fastream-II is inspired by the primal-dual schema from Garg and Konemann [67]. However, previous approximation solutions to the MCMF problem generally assume flow conservation on nodes—i.e., incoming commodities and outgoing commodities are exactly equal in amount. This is not true in P2P streaming, where peers can reproduce whatever commodities they receive—i.e., flow conservation does not hold. In addition, the MCMF problem considers only the capacities on edges, whereas in P2P streaming, the capacities actually exist on nodes instead of edges. This distinction (for the MADPS problem) further requires optimal flow scheduling among edges departing from the same node [69]. All these differences make the MADPS problem more complex than the MCMF problem. Our work tackles these complexities and achieves a solution with near-optimal performance bound.

2.3 Strategies on Minimizing Churn-Induced Delays

On the following, we introduce the related works on reducing churn-induced delay. Previous works on IPTV show that the next channel that a user may watch can be predicted using probability analysis based on user's past channel switching patterns [70–73]. 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 may switch to an adjacent channel in the channel list when he or she is seeking some interesting channel in sequence. Thus, several papers, such as [70, 71, 74], on multicast IPTV systems propose to send contents of the predicted next channels in parallel with the currently viewed channel. Should the user switch to one of the predicted channels, he or she can immediately watch the next channel without delay. However, such a method is bandwidth consuming in transmitting multiple streams, which is not practical for current P2P streaming systems due to limited upload and download bandwidth.

The channel-switching delay problem has high similarity with the segment-seeking delay problem in video-on-demand (VoD) systems, where users observe a delay when they're browsing the channel for a segment that they are interested in citeguo06, pengzhu10. Segment-seeking delay is defined as the interval between the time when a segment is requested and the time when the segment is ready for playback [75]. Existing studies on video-on-demand (VoD) [75,76] have resulted in prefetching algorithms with segment prediction, which can be effective for reducing the seeking delay. A prefetching scheme typically streams one or multiple predicted segments while the current segment is being played [75]. For example, in [75], He *et al.* present 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 [9]. However, such methods cannot be extended to live channel applications, where prefetching of live video content is not possible.

The problem of reducing churn-induced delay in P2P streaming has close relationship to the problem of improving the churn resilience, because a resilient system generally has less streaming interruptions [77–80]. In [77], Hernandez-Serrano *et al.* proposed two protocols, TrebleCast and TrebleCast*, to implement reliable overlay networks resilient to network churn. The major strategy behind these two protocols is to move peers with higher expected lifetime to the center of the overlay while other peers will remain on the outside layers. In the presence of churn, these more reliable layers on the overlay center will be called to deliver the data content. However, maintenance of such structured overlay construction is cost-expensive in a dynamic network. For example, the expected lifetime of peers will change with their uptime, which will lead to frequent overlay maintenance and message exchange. In addition, the resource search time may take a long time if a large number of layers exist in the overlay [77]. Moreover, there may be a greater load on the core layers if the inner "reliable" layers are small [77].

In [8], Wu proposeed a strategy, called view-upload decoupling (VUD), to reduce the channel churn. The VUD-based P2P streaming approach decouples peers' viewing from their content uploading. In other words, what a peer uploads is independent of what it views [13]. 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 is that VUD costs more bandwidth overhead. To tackle this problem, Wu *et al.* [13] proposed a substream-swarming enhancement. Specifically, each swarm only distributes a small portion of the stream, called a substream, to viewers. The viewers need 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 systems, where peers deliver the same channel content that they are viewing. As a churn-resilient scheme, the channel-switching behavior of a peer in VUD will not influence its downstream peers. Yet, VUD can neither reduce the delay consumed at the peer that is switching the channel, nor the delay from the node churn.

Instead of mitigating one type of delay, our strategy focuses on retrenching the connection time for all cases of churn-induced delays. Additionally, considering the promising advantages of VUD, NAP is capable of integrating our scheme with VUD to obtain better performance.

Chapter 3

Preliminaries and Objectives

In this chapter, we first discuss the general system model and related assumptions, which are extensively used in our problem space. For specific models and assumptions that are unique to one problem, we will discuss them in the corresponding chapters. We then describe the preliminaries about the approximation algorithms and primal-dual related theorems. We also present the objective problems in this section.

3.1 General Network Model and Assumptions

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 chapter, 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, 81].

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* [21]. 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 [21].

Current works on P2P streaming can be broadly classified into two classes: (1) tree-based overlays, and (2) mesh-based overlays [9, 18–22]. Recent studies have shown that the mesh-based approach consistently exhibits a superior performance over the tree-based approach [19, 23]. The major advantages of mesh-based systems are easy maintenance and inherent robustness in high-churn P2P environment [9]. Motivated by these promising advantages, we study the problems under the mesh-based model. Although our algorithms are developed with a mesh paradigm, they are ready to fit the multiple tree-based model after simple modifications as well. To achieve a tractable theoretical analysis, we assume no network dynamics in the first stage of algorithm design. The method will be most suitable for the scenario where a service provider deploys a set-top box at viewers' homes. In that case, even when a viewer turns off the TV, the set-top box can still contribute its bandwidth to other viewers. Although this assumption is strong in practical P2P applications, the value of this work lies in the theoretical framework and analysis, which sheds light on the practical design. Considering a dynamic network in real implementation, we then study churn-resilient scheme at the last chapter of this dissertation. To reduce the complexity of the problem, we focus only on minimizing the communication delay.

Furthermore, to obtain a tractable theoretical bound, we assume a network model with a symmetric graph and satisfying the triangle inequality in the design of Faststream-I. Then, in Faststream-II and Faststream-III, we remove those assumptions in modeling the network systems.

3.2 Approximation Algorithms

Approximation algorithms are algorithms devised to find near-optimal solutions to optimization problems. Approximation algorithms are widely used when it is unlikely to find an efficient polynomial time exact algorithms [82]. For example, approximation algorithms are often utilized to solve NP-hard optimization problems, which have no known solutions in polynomial time. Sometimes, approximation algorithms are used to improve the speed of solutions even if there is a known polynomial time exact algorithms. In this case, the exact algorithms are normally too expensive to run due to the large input size.

Although approximation algorithms do not provide the exact solutions to the problem, they can guarantee a worst-case performance bound. As we know, heuristics can also provide reasonably good solutions with reasonably fast running time. However, they are empirically established without any theoretical guarantee on the worst-case performance and running time. In contrast to heuristics, approximation algorithms are generated with provable solution quality and provable running time bounds.

Since performance bound is the essence of approximation algorithm, we make the follow definitions about the approximation factor.

Definition 1. *For a minimization problem, the approximation factor is the ratio of worst-*

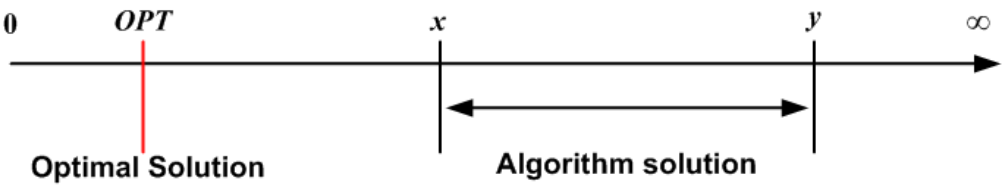


Figure 3.1: Approximation factor.

case performance to the optimal value.

Figure 3.1 illustrates an example of this definition, where OPT is the exact optimal solution, y is the worst-case solution of the approximation algorithm, and x represents the lower-bound that an approximate solution could span, i.e., $OPT \leq x \leq y$. The actual solution that the approximation algorithm provides should be within $[x, y]$. Since it is a minimization problem and y provides the upper-bound, we can see the bound of this approximation algorithm, or called approximation factor, denoted by ρ, can thus be expressed by

$$\rho = \frac{y}{OPT}.$$

For a maximization problem, we define the approximation factor as follows.

Definition 2. *For a maximization problem, the approximation factor is the ratio of the optimal value to the worst-case performance.*

In this dissertation, we focus on approximate algorithms for minimum streaming delay problems because we target time-critical P2P applications (e.g., video conferencing, or cloud computing), for which assured bounds on performance and running time are more desirable than heuristics. In addition, the analytical foundation that is necessary for developing approximate algorithms can contribute to a greater understanding of the problem and can provide deeper insights on designing efficient algorithms, be they approximate or heuristic.

3.3 Primal and Dual

Linear programming (LP) is widely used to formulate such problems that can be expressed by optimizing (i.e., minimizing or maximizing) a linear function subject to linear inequality constraints [82, 83]. We call the function being optimized as *objective function*. Every linear programming problem can be referred to as a *primal* problem, or simply called primal. A primal problem can be converted to a *dual* problem, or called dual, by a mechanic procedure. Since primal and dual are in a "mirror" relation, dual problem can also be converted to primal problem. Table 3.1 provides two examples of primal and dual conversion.

Primal	**Dual**
$\begin{aligned} \min \quad & \mathbf{A}^{\mathrm{T}}\mathbf{X} \\ \text{subject to} \quad & \mathbf{B}\mathbf{X} \geq \mathbf{C} \\ & \mathbf{X} \geq \mathbf{0} \end{aligned}$	$\begin{aligned} \max \quad & \mathbf{C}^{\mathrm{T}}\mathbf{Y} \\ \text{subject to} \quad & \mathbf{B}^{\mathrm{T}}\mathbf{Y} \leq \mathbf{A} \\ & \mathbf{Y} \geq \mathbf{0} \end{aligned}$
$\begin{aligned} \max \quad & \mathbf{A}^{\mathrm{T}}\mathbf{X} \\ \text{subject to} \quad & \mathbf{B}\mathbf{X} \leq \mathbf{C} \\ & \mathbf{X} \geq \mathbf{0} \end{aligned}$	$\begin{aligned} \min \quad & \mathbf{C}^{\mathrm{T}}\mathbf{Y} \\ \text{subject to} \quad & \mathbf{B}^{\mathrm{T}}\mathbf{Y} \geq \mathbf{A} \\ & \mathbf{Y} \geq \mathbf{0} \end{aligned}$

Table 3.1: Primal-Dual Conversion Mechanism

Dual problem is also formulated in a linear programming framework. If primal is a minimization/maximization problem, dual is a maximization/minimization problem correspondingly, and vice versa. Although the objective functions of primal and dual are different, they share the same optimal value. This rule is the central theorem of linear programming, called *LP-duality theorem* [82].

Theorem 1. (LP-duality theorem) [82] *The primal has finite optimum iff its dual has finite optimum. Moreover, given the same problems as in Table 3.1, if* $\mathbf{X}^*$ *and* $\mathbf{Y}^*$ *are optimal solutions for the primal and dual problems, respectively, we have*

$$\mathbf{A}^{\mathrm{T}}\mathbf{X}^* = \mathbf{C}^{\mathrm{T}}\mathbf{Y}^*.$$

Figure 3.2 shows the examples of primal-dual relation when the primal problem is a minimization problem and maximization problem, respectively. Given the primal problem is a minimization/maximization problem, any feasible dual solution provides a lower/upper bound on the optimum of the primal problem. This leads to another import theorem in LP, called *Weak duality theorem*.

Theorem 2. (Weak duality theorem) [82] *Given the primal problem as the minimization problem in Table 3.1, if* $\mathbf{X}$ *and* $\mathbf{Y}$ *are feasible solutions for the primal and dual problems, respectively, we have*

$$\mathbf{A}^{\mathrm{T}}\mathbf{X} \geq \mathbf{C}^{\mathrm{T}}\mathbf{Y}.$$

Similarly, given the primal problem as the maximization problem in Table 3.1, we have

$$\mathbf{A}^{\mathrm{T}}\mathbf{X} \leq \mathbf{C}^{\mathrm{T}}\mathbf{Y}.$$

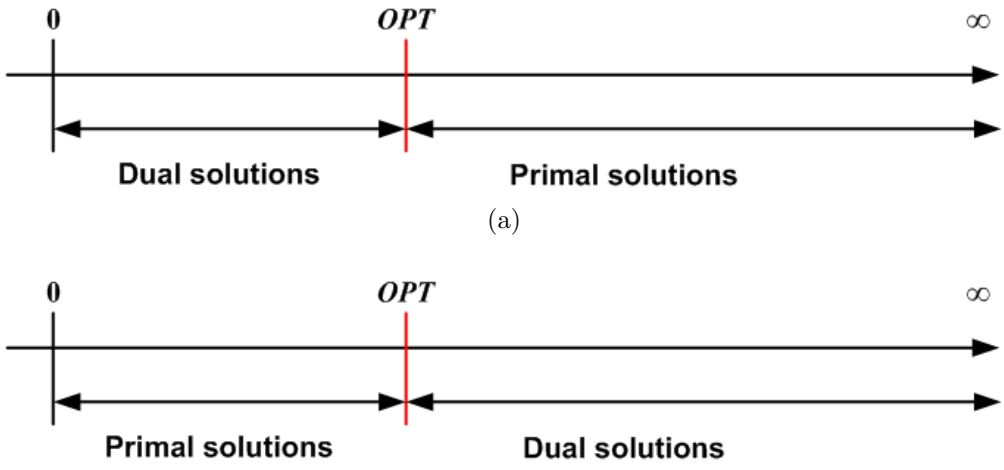


Figure 3.2: Primal and dual share the same optimal value: (a) Primal is minimization problem; (b) Primal is maximization problem.

We do not provide detailed proof of weak duality theorem here. For readers who are interested in this, it can be found in [82].

LP-duality theorem provides the foothold for algorithms that provide exact solutions. In contrast, the weak duality theorem generally suffices in approximation algorithms [82].

3.4 Objectives

In this dissertation, we focus on optimizing delays in P2P live streaming systems from four aspects: (1) minimizing the maximum end-to-end delay in P2P streaming; (2) minimizing average end-to-end delay in P2P streaming; (3) minimizing average delay in multi-channel P2P streaming; (4) reducing churn-induced delays. In the following, we specify the objectives to be solved in the dissertation.

3.4.1 Minimizing Maximum Delay in P2P Streaming

We call the stream from the source to one receiver j as the P2P *unicast flow* to j. A P2P unicast flow U_j to peer j may consist of streams from multiple P2P paths, called *fractional flows* [35]. Each fractional flow $p \in U_j$ has the arrival latency $l(p)$ from the source to receiver, i.e., *end-to-end delay*, where $l(p) = \sum_{(i,j)\in p} d_{(i,j)}$. The latency of a unicast flow U_j can be defined as the maximum latency among its fractional flows to j, i.e., $\max_{p\in U_j} l(p)$. To stream multimedia content to multiple receivers, we can envision multiple unicast flows from the source to receivers.

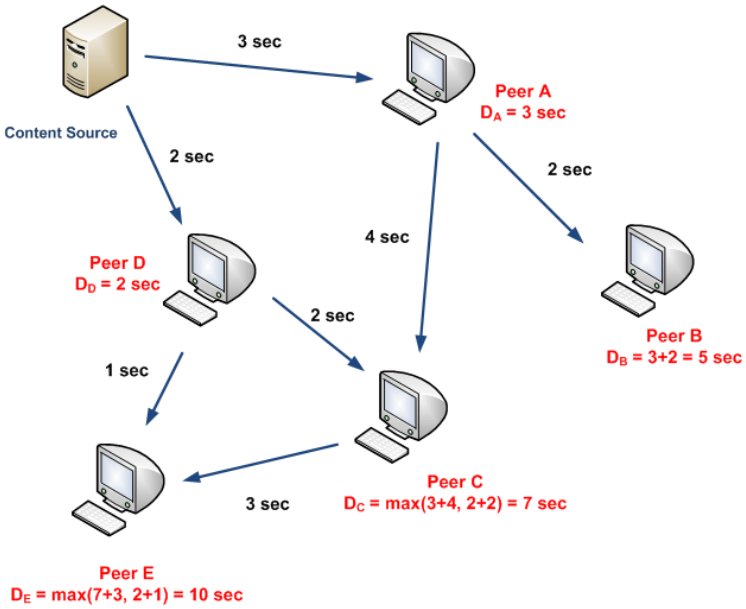


Figure 3.3: The maximum delay in this network is 10 seconds, which happens on peer E.

The maximum delay in P2P streaming can then be defined as the maximum latency among all the fractional flows in source to all receivers. Denote the maximum delay by D_max. It can be expressed by

$$D_{max} = \max_{p \in \bigcup_{j \in R} U_j} l(p). \tag{3.1}$$

In Figure 3.3, we illustrate the definition of the maximum delay in the network. For unicast flow U_j to viewer j, we denote the maximum delay of this unicast flow as D_j and source node as S. For example, node C in total receives two fractional flows via path $< S, A, C >$ and $< S, D, C >$, respectively. $< S, A, C >$ has an end-to-end delay of 7 seconds, and $< S, D, C >$ has an end-to-end delay of 4 seconds. Thus, we have $D_j = 7$ seconds. Comparing all the unicast flows, we find the maximum delay in the P2P networks is 10 seconds, which happens on peer E.

Our first objective is to minimize D_{max} in the network. We call it MDPS problem.

3.4.2 Minimizing Average Delay in P2P Streaming

Each fractional flow is associated with a streaming rate. We define the average end-to-end delay of the unicast flow U_j as the weighted average of end-to-end latencies of all its fractional flows, where the weight is the portion of fractional flow rate to the total streaming rate. Denote $f(p)$ as the streaming rate of fractional flow p. For viewer j, the weighted

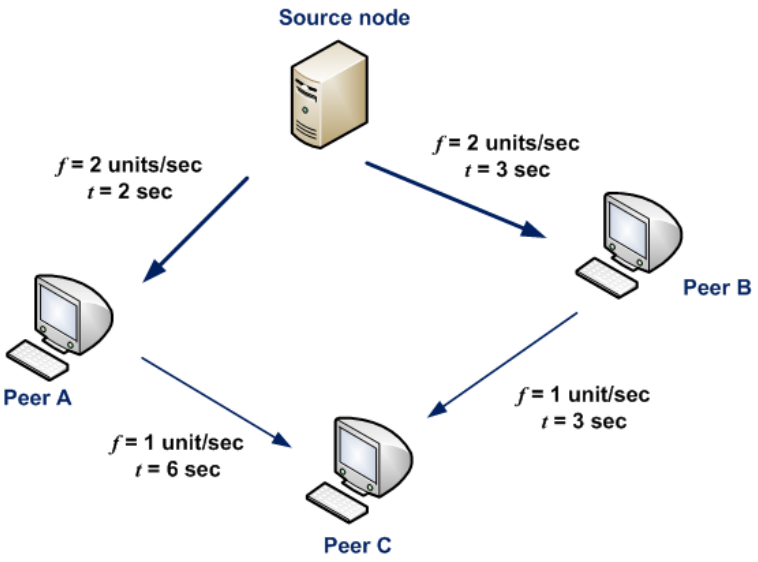


Figure 3.4: The average delay in the network is 4 seconds.

average of end-to-end latencies can be expressed by

$$\frac{1}{d_j}\sum_{p\in U_j} l(p)f(p),$$

where d_j is the demanded streaming rate on viewer j.

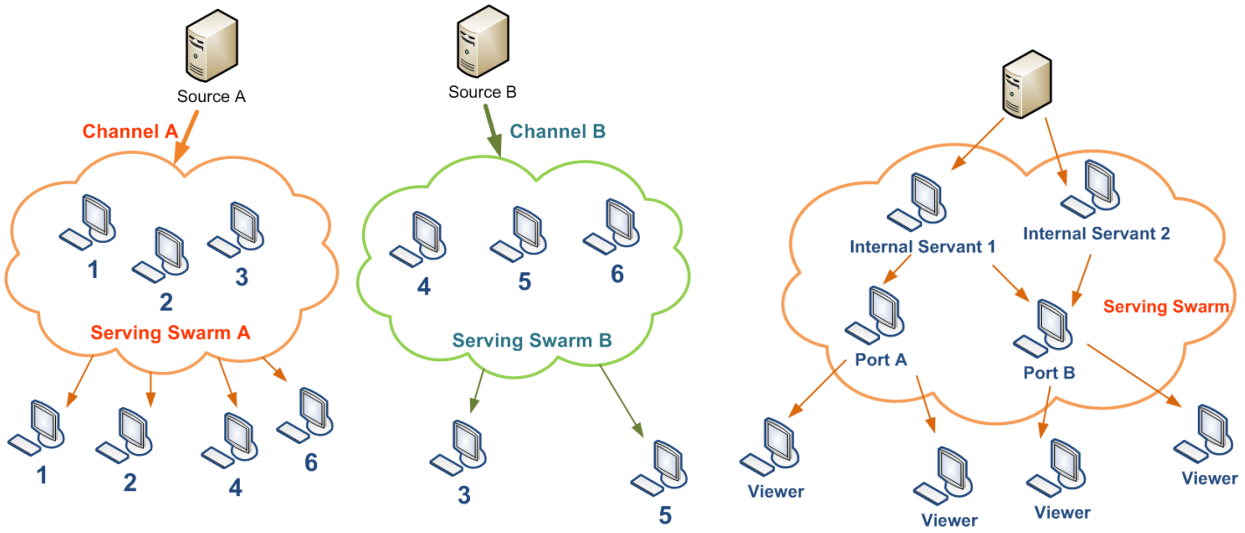
For all viewers, the weighted average of end-to-end latencies can be expressed by

$$\frac{1}{\sum_{j\in R} d_j}\sum_{p\in P} l(p)f(p),$$

where $P = \bigcup_{j\in R} U_j$.

For example, in Figure 3.4, the demanded streaming rate on each viewer is 2 units/second. Here, f represents the streaming rate on the associated link, and t means the link latency. It is straightforward to find the latency and streaming rate data for node A and node B. For node C, it has two fractional flows via paths $< S, A, C >$ and $< S, B, C >$, respectively. $< S, A, C >$ has an end-to-end delay of 8 seconds and streaming rate of 1 unit/second. $< S, B, C >$ has an end-to-end delay of 6 seconds and streaming rate of 1 unit/second. Thus, the average delay can be carried out by $\frac{1}{3\times 2}(2\times 2+2\times 3+8\times 1+6\times 1) = 4$ seconds.

Our second objective is to minimize average delay in the network. We call it MADPS problem.



(a) (b)

Figure 3.5: View-upload decoupling: (a) VUD streaming model; (b) Components inside a serving swarm.

3.4.3 Minimizing Average Delay in Multi-channel P2P Streaming

Existing P2P streaming applications, such as PPStream and PPLive, can support more than hundreds of channels. In multi-channel scenario, viewers frequently switch between channels. Such channel switching behaviors bring extra challenges to minimizing the streaming delay. Suppose we have the solutions to minimize the delay in single channel scenario. They can not be directly utilized in multichannel scenario because every channel switching behavior will lead to a change in the overlay topology and thus need to rerun the optimization process.

To reduce the influence from the channel churn, Wu *et al.* proposed a view-upload decoupling (or VUD) model to improve the streaming qualities on all channels [8]. In VUD model, what a peer uploads is independent of what it views and subscribes [13]. We call the set of peers uploading data for channel A as the *serving swarm* A. Figure 3.5a presents an example of VUD model, where two channel source nodes are distributing the channel contents to two serving swarms, respectively. Once a viewer switches to channel A, it contacts serving swarm A and then receives the stream from nodes in serving swarm A. In VUD model, nodes play two roles: viewer and uploader inside the serving swarm. We can see from the example that node 6 serves in swarm B but watches channel A. When node 6 switches to view other channel, it remains serving channel B. Such design greatly improves the robustness in the high-channel-churn environments and is able to flexibly assign more peers to assist the channels with less bandwidth resources. Figure 3.5b zooms inside the infrastructure of a serving swarm. We call the peers that stream the data directly to the viewers as *ports*. Besides ports, there is another category of peers inside the serving swarm. They forward the content directly or indirectly to the ports. We call them *internal servants*.

Because viewers may switch channels dynamically, minimizing the end-to-end delay is less meaningful. Based on the sparkling advantages of VUD model, our third objective is to minimize the weighted average streaming delay from the source nodes to the ports. We call it MSPDS problem. Towards a general solution, we assume each channel streams from a distinctive source node. In reality, channels may share the source node. We can easily handle this case by viewing some source nodes as "avatars." Our strategy provides a general solution.

For simplicity, we also assume that internal servants only upload data to peers inside the serving swarm while ports only feed data to viewers. In this dissertation, we do not work on the methodology on how to select the ports from the serving swarm. Generally, we can employ existing solutions in terms of superior peer selection as in [40, 42, 84].

3.4.4 Reducing Churn-induced Delays in P2P Streaming

Nodes may join or leave the network dynamically. Peer departure can lead to a streaming interruption to its downstream peers. In such case, delay happens during the restoration of streaming connections [13]. We call such delay *recovery delay*. Another type of churn is *channel churn*, which represents the event of channel switching in a multi-channel P2P streaming system. It has been observed that channel churn is much more frequent than node churn [8], and can cause severe instability to the system [85]. In a multi-channel P2P streaming system, the video content of each channel is distributed by a different swarm of peers. Similar to 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. The time consumed in connecting to the new swarm is called *channel switching delay*. Since recovery delay and channel switching delay both arise from the network churn, we generally call them *churn-induced delays*.

Towards improving the user expedience in terms of such delays, the fourth objective in this dissertation is about reducing the churn-induced delays, including recovery delay and channel switching delay.

Chapter 4

Minimizing Maximum P2P Streaming Delay

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

4.1 Problem Formulation

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

4.1.1 The Minimum-Delay P2P Streaming Problem

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 $d_{(i,j)}$. 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. We consider a peer-to-peer streaming session to originate from a single source node S to a set of receivers R, where $V = \{S\} \cup R$. 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.

From Section 3.4.1, we know the maximum delay in P2P streaming is defined as the maximum latency of all unicast flows. We now define the problem formally:

Definition 3. ***Minimum-Delay P2P Streaming Problem (MDPS problem):*** *Given the constraints on link delays and node capacities, 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:

$$\min\ t \tag{4.1}$$

subject to

$$\sum_{(i,j)} d_{(i,j)} x^r_{ijm} \leq t, \forall (i,j) \in E, \forall r \in R, \forall m \tag{4.2}$$

$$x^r_{ijm} \in \{0,1\}, \forall (i,j) \in E, \forall r \in R, \forall m \tag{4.3}$$

$$x^r_{ijm} \geq f^r_{ijm}/s, \forall (i,j) \in E, \forall r \in R, \forall m \tag{4.4}$$

$$s \geq f^r_{ijm} \geq 0, \forall (i,j) \in E, \forall r \in R, \forall m \tag{4.5}$$

$$f^r_{ijm} - f^r_{jim} = b^r_{im}, \forall (i,j) \in E, \forall r \in R, \forall m \tag{4.6}$$

$$\sum_{m=1}^{s} \sum_{r:r\in R} f^r_{ijm} \leq y_{ij}, \forall (i,j) \in E \tag{4.7}$$

$$\sum_{j:(i,j)\in E} y_{ij} \leq O_i, \forall i \in V \tag{4.8}$$

$$\sum_{j:(j,i)\in E} y_{ji} \leq I_i, \forall i \in V \tag{4.9}$$

$$0 < m \leq s, \tag{4.10}$$

where

$$b^r_{im} \begin{cases} \geq 0 & \text{if } i = S, \\ \leq 0 & \text{if } i = r, \\ = 0 & \text{otherwise,} \end{cases} \tag{4.11}$$

$$\sum_m b^r_{im} = \begin{cases} s & \text{if } i = S, \\ -s & \text{if } i = r, \\ 0 & \text{otherwise.} \end{cases} \tag{4.12}$$

In this integer programming (IP) expression, t denotes the streaming delay of the fractional flow; x^r_{ijm} 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^r_{ijm} 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.,*

$$\sum_{i \in V} O_i \geq (|V| - 1) \times s. \tag{4.13}$$

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 (4.13) holds. In addition, we presume that the download capacity $I_i \geq s, \forall i \in V$ for a smooth playback at receivers.

4.1.2 Hardness

Theorem 3. *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 [86, 87]. 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 awakened, 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 [86].

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 (4.8), (4.9), (4.12). Thus, we complete the proof. □

4.2 Approximation Algorithm

4.2.1 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 [64] by Könemann *et al.* 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 [88, 89]. Therefore, we will not focus on this scenario in this work.

In the rest of this section, we discuss the details of Fastream-I and derive its performance bound.

4.2.2 Centralized Approximation Algorithm

Streaming Backbone Construction

The first step of Fastream-I 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:

$$C(V') = \sum_{i \in V'} O_i - (|V'| - 1) \times s. \tag{4.14}$$

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 $E_\gamma(v_i, V)$ [64].

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 its 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.:

$$u_i = \text{argmax}_{v \in W_i^\gamma} C(E_\gamma(v, W_i^{3\gamma})). \tag{4.15}$$

Now, we have $U_i = E_{3\gamma}(u_i, W_i^{3\gamma}) \cup \{u_i\}$, where we define u_i as the representative node of cluster U_i. 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 $W_i^{3\gamma} \neq \emptyset, \forall 1 \le i \le k$. Figure 4.1 illustrates an example of clustering steps. Figure 4.1a presents the nodes before clustering procedures. In Figure 4.1b, we pick source node as the first center; then, mark all nodes within γ radius of u_1 as inactive nodes and 3γ radius of u_1 as covered nodes. In Figure 4.1c, we pick node whose γ-radius active neighbors cover the most residual streaming capacity as u_2; then, mark all nodes within γ radius of u_2 as inactive nodes and 3γ radius of u_2 as covered nodes. In Figure 4.1d, we pick node whose γ-radius active neighbors cover the most residual streaming capacity as u_3. After that, we mark all nodes within γ radius of u_3 as inactive nodes and 3γ radius of u_3 as covered nodes. Figure 4.2 presents the clustering result, where three clusters are generated.

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., $C(U_i) \ge C(U_j), \forall 1 < i < j \le k$. 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. *If a peer u is 2γ-covered by u_i but not in cluster U_i, u resides either in cluster U_1, or in cluster U_j with $C(E_\gamma(u_j, V)) \ge C(E_\gamma(u_i, V))$.*

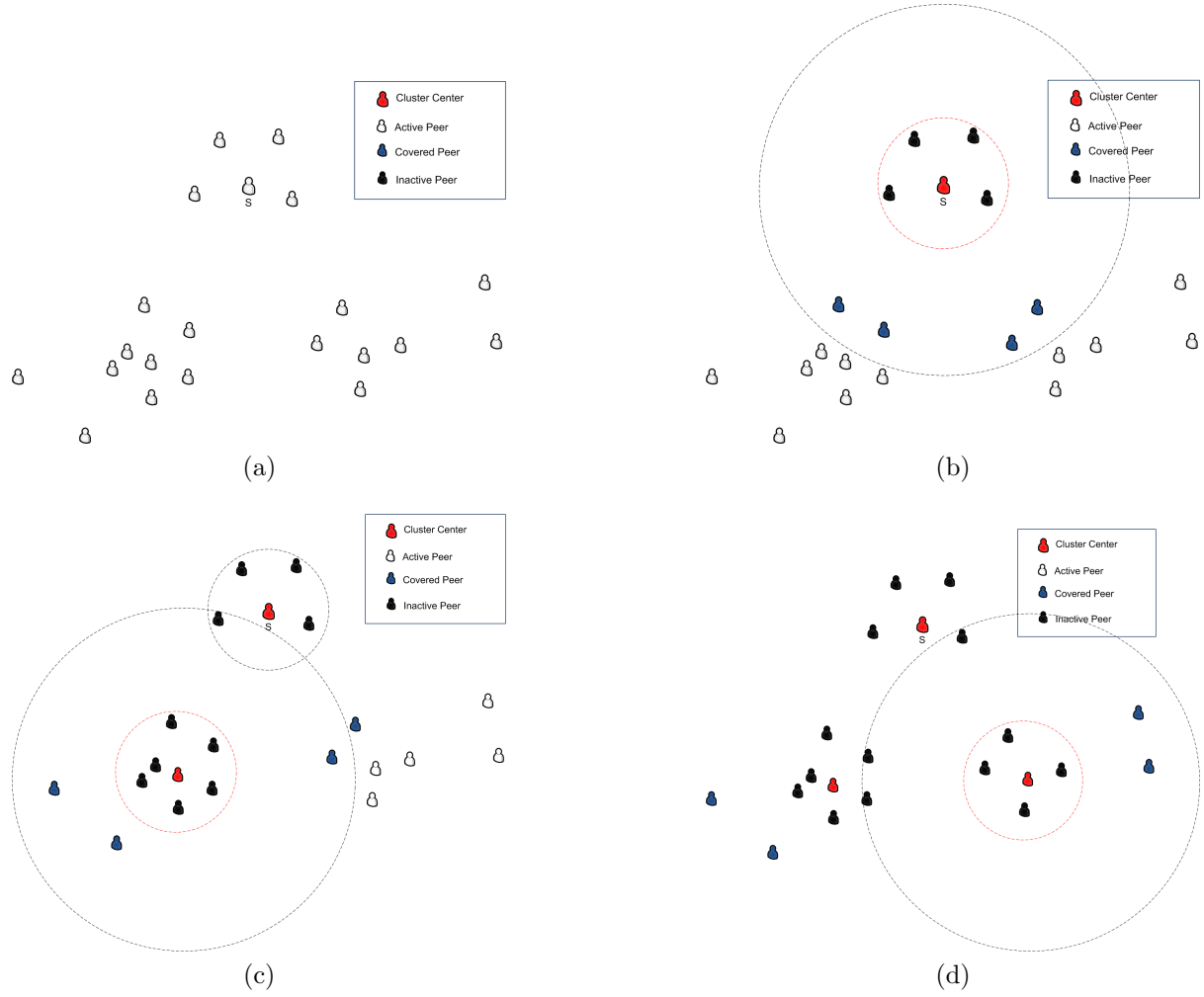


Figure 4.1: An example of clustering procedures: (a) Nodes before clustering procedures; (b) Pick source node as the first center; Mark all nodes within γ radius of u_1 as inactive nodes and 3γ radius of u_1 as covered nodes; (c) Pick node whose γ-radius active neighbors cover the most residual streaming capacity as u_2; Mark all nodes within γ radius of u_2 as inactive nodes and 3γ radius of u_2 as covered nodes; (d) Pick node whose γ-radius active neighbors cover the most residual streaming capacity as u_3; Mark all nodes within γ radius of u_3 as inactive nodes and 3γ radius of u_3 as covered nodes.

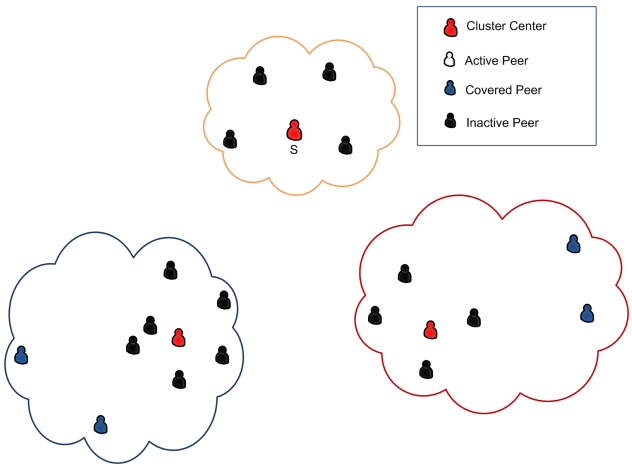


Figure 4.2: Result after clustering procedures.

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

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

Regional Streaming Mesh Construction

In this section, we show the steps to create 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

```
for i = 1 to k do
  for j = 1 to |U_i| do
    repeat
      if j is the peer with largest uploading capacity then
        Hold j for an uplink connection from other cluster
      else
        x = argmax_{v∈U_i & v has remaining bandwidth} C({v})
        if x is fully served then
          Connect j to peer x
        else
          Hold j for external connection
        end if
      end if
    until peer j is fully served or on hold
  end for
end for
```

Complete Streaming Mesh Construction

To complete the final mesh construction, we replace virtual links between representatives with 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 4.3 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 $d_{(u_i^*,u)} \leq \gamma\}$;
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.

Corollary 2. *U_1^* is a subset of U_1, i.e., $U_1^* \subseteq U_1$.*

Lemma 2. *Suppose U_i^* and U_j intersects, i.e., $U_i^* \cap U_j \neq \emptyset$. Then, if $U_i^* \setminus U_j \neq \emptyset$, 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. We prove the lemma by analyzing the possible distance between u_j and w, where $w \in U_i^* \cap U_j$.

1) In the case of peer w in $U_i^* \cap U_j$ is within the γ radius of u_j, i.e., $d_{(w,u_j)} \leq \gamma$, we begin the proof by finding the first cluster that covers any peer in U_i^* during the run time of Algorithm 1. We denote such cluster as U_m. Apparently, $C(U_m) \geq C(U_i^*)$; otherwise, U_i^* itself will form a cluster itself during the run time.

Next, we prove U_m covers at least one peer in $U_i^* \setminus U_j$. If u_i^* resides within the γ radius of u_j, any $u \in U_i^* \setminus U_j$ will be 2γ-covered by u_j according to the triangle inequality; otherwise, u_i^* is 2γ-covered by u_j. Thus, at least one 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_x covering u , where $C(E_\gamma(u_x, V)) \geq C(E_\gamma(u_j, V))$ and $x \neq j$. (Because $U_1^* \subseteq U_1$, u_x cannot be S.) Since u_m is the first cluster that covers at least one peer in U_i^*, $C(E_\gamma(u_m, V)) \geq C(E_\gamma(u_x, V))$, which makes it impossible that U_m only covers no peer in $U_i^* \setminus U_j$.

For the sake of contradiction, we assume u_j is u_m. Then, every peer 2γ-covered by u_j cannot 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 $C(U_m) \geq C(U_j) \geq C(U_i^*)$.

2) 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.

3) 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:*

$$\sum_{i=1}^{j} C(U_i^*) \leq \sum_{i=1}^{j} C(U_i), \forall 1 \leq j \leq q. \tag{4.16}$$

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

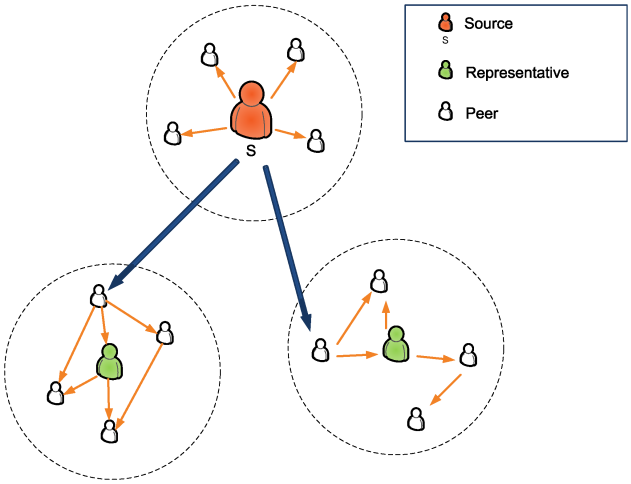


Figure 4.3: Cluster-based streaming mesh.

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 3 from Lemma 3:

Corollary 3. *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:

$$H_{T_B} \leq H_{T_B^*} \leq \frac{\text{OPT}}{\gamma} = \frac{\text{OPT}}{t'/\sqrt{\log n}} = O(\sqrt{\log n}). \tag{4.17}$$

□

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 other words, 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

$$\begin{aligned}
\zeta_X &\leq H_{U_1} + H_{U_{x_1}} + \sum_{i=1}^{b} \log_2 \left(C\left(U_{y_i}\right) - 1 \right) \\
&= H_{U_1} + H_{U_{x_1}} + \log_2 \left(\prod_{i=1}^{b} \left(C\left(U_{y_i}\right) - 1 \right) \right).
\end{aligned} \tag{4.18}$$

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

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

Thus, we have

$$C(U_1) \cdot \prod_{i=1}^{b-1} C(U_{y_i}) \leq n. \tag{4.19}$$

Replacing Equation (4.19) into (4.18), we have

$$\begin{aligned} \zeta_X &\leq H_{U_1} + H_{U_{x_1}} + \log_2 \left(\prod_{i=1}^{b} C(U_{y_i})\right) \\ &= H_{U_1} + H_{U_{x_1}} + \log_2 \left(C(U_1) \cdot \prod_{i=1}^{b-1} C(U_{y_i})\right) \\ &\quad + \log_2 \left(C(U_{y_b}) / C(U_1)\right) \\ &\leq H_{U_1} + H_{U_{x_1}} + \log_2 n + \log_2 \left(C(U_{y_b}) / C(U_1)\right) \\ &\leq 4 \log_2 n \\ &= O(\log n). \end{aligned} \tag{4.20}$$

Thus, the lemma follows. □

Theorem 4. *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 5. *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 (4.18). 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. As a result, we can carry out

$$\begin{aligned}
\zeta_X &\leq H_{U_1} + H_{U_{x_1}} + \sum_{i=1}^{b} \log_2 \left(C\left(U_{y_i}\right) \times \frac{2s}{\min(O_i)} - 1 \right) \\
&\leq H_{U_1} + H_{U_{x_1}} + \sum_{i=1}^{b} \log_2 \left(C\left(U_{y_i}\right) \times \frac{2s}{\min(O_i)} \right) \\
&= H_{U_1} + H_{U_{x_1}} + \log_2 \left(\prod_{i=1}^{b} C\left(U_{y_i}\right) \right) + b \cdot \log_2 \frac{2s}{\min(O_i)} \\
&= H_{U_1} + H_{U_{x_1}} + \log_2 \left(\prod_{i=1}^{b} C\left(U_{y_i}\right) \right) + \log_2 n \times \log_2 s. \qquad (4.21)
\end{aligned}$$

Similar to the deduction of Equation (4.20), we have

$$\begin{aligned}
\zeta_X &\leq (4 + \log_2 s) \cdot \log_2 n \\
&= O(\log n). \qquad (4.22)
\end{aligned}$$

Thus, the theorem follows.

□

4.2.3 Distributed Algorithm

The centralized algorithm described in Section 4.2.2 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 Fastream-I 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 the buffering time 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.

4.3 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 [21, 81]. Based on this, we set the upload capacity among peers as shown in Table 4.1. In this simulation study, we compare our algorithm with two other recently proposed algorithms: a heuristic approach [21] and a LP-based approach [35]. The heuristic in [21] 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 [21]. The LP-based approach in [35] applies several linear programming techniques to obtain an optimal average delay, such as Lagrangian relaxation and subgradient algorithm. Please note, to reduce computational costs, [35] restricts the potential connections for each peer. This actually lowers 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 chapter.

To evaluate the algorithm performance, we define three metrics, including *average end-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 4.4 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

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

Table 4.1: Upload Capacity Distribution

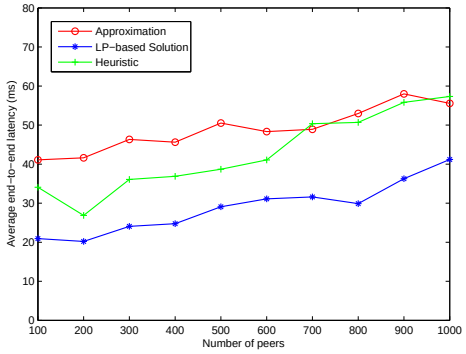


Figure 4.4: Average end-to-end latency

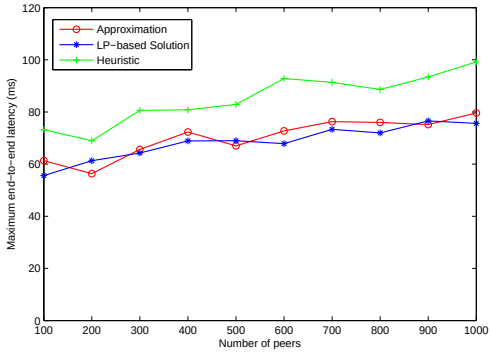


Figure 4.5: Maximum end-to-end latency

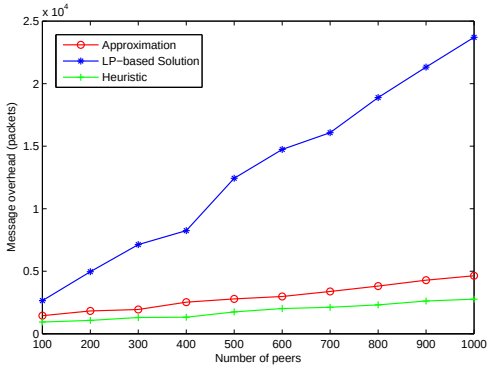


Figure 4.6: Message overhead

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 4.5 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 Fastream-I ensures good streaming performance with an approximation bound.

Figure 4.6 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 Fastream-I 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.

4.4 Summary

In this chapter, we focused on 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. For a tractable solution, we assume a symmetric network observing the triangle property. Future study could work on relieving such assumptions on the MDPS problem.

Chapter 5

Minimizing Average P2P Streaming Delay

In this chapter, we describe the problem of minimizing average delay in P2P streaming, i.e. MADPS problem. We formulate MADPS problem in linear programming framework. Then we devise our proposed approximation algorithm – Fastream-II based on primal-dual schema. Next, we justify its near-optimum performance assurance. Simulations are implemented and the results are presented at the end of this chapter.

5.1 Problem Formulation

In this section, we formally state the minimum average end-to-end delay P2P streaming (MADPS) problem and present the problem in linear programming (LP) framework.

5.1.1 Preliminaries and Modeling

In contrast to the solution in MDPS problem, we remove the assumptions on symmetric network and triangle property. We consider a directed graph. Each overlay link $(i, j) \in E$ is associated with a communication delay l_{ij}, where l_{ij} is not necessarily equal to l_{ji}.

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. In practical applications, receivers may pay for services of different streaming qualities, e.g., 720i/p and 1080i/p, which leads to different streaming rates correspondingly. Suppose peer j selects a service that has a constant streaming rate of d_j units/second. We denote f_{ij} as the rate at which peer i streams to peer j. If peer j receives the aggregated non-identical streams at d_j units/second from its

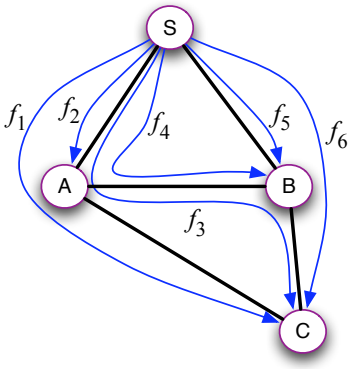


Figure 5.1: *A P2P network with 4 nodes S, A, B, C. Node S is the source and set of receivers $R = \{A, B, C\}$. A node can receive flow via multiple paths; for example, nodes C receives 3 flows f_1, f_3 and f_6 via paths $\langle S, A, C\rangle, \langle S, A, B, C\rangle$ and $\langle S, B, C\rangle$ respectively. There can be multiple flows through an edge to the same destination; for example, flow f_1 and f_3 to receiver C through (S, A). There are two other flows f_2 and f_4 through (S, A) to receivers A and B, respectively. We can observe these flows actually originate from one merged flow from S to A, which is reproduced (replicated) at node A again. Thus, the actual flow through link (S, A) is* $\max(f_2, f_4, f_1 + f_3)$.

parents, we call peer j as *fully served* [21]. Mathematically, the fully served requirement of peer j can be expressed as $\sum_{i:i\in L_j} f_{ij} = d_j$, where L_j is the set of parents of peer j.

We call the stream from the source to one receiver j as the *P2P unicast flow* to j. Each P2P unicast flow U_j may consist of streams from multiple P2P paths, called *fractional flows* [35]. Each fractional flow $p \in U_j$ has the arrival latency $l(p)$ from the source to receiver, i.e., *end-to-end delay*, where $l(p) = \sum_{(i,j)\in p} l_{ij}$. We define the average end-to-end delay of the unicast flow U_j as the weighted average of end-to-end latencies of all its fractional flows, where the weight is the portion of fractional flow rate to the total streaming rate. Denote $f(p)$ as the streaming rate of fractional flow p. For viewer j, the weighted average of end-to-end latencies can be expressed by

$$\frac{1}{d_j} \sum_{p\in U_j} l(p) f(p).$$

To stream multimedia content to multiple receivers, we can envision multiple unicast flows from the source to receivers. Thus, the *average end-to-end delay in P2P streaming* is defined as the weighted average latency of all fractional flows to all receivers, which can be described by

$$\frac{1}{\sum_{j\in R} d_j} \sum_{p\in P} l(p)f(p), \tag{5.1}$$

where $P = \bigcup_{j\in R} U_j$. Since the term $\sum_{j\in R} d_j$ has no effect on the optimal solution, i.e., the solution that minimizes (5.1) also minimizes $\sum_{p\in P} l(p)f(p)$, we will focus on minimizing $\sum_{p\in P} l(p)f(p)$. It is easy see that removal of the term $\sum_{j\in R} d_j$ also preserves the approximation factor. For ease of presentation, we simply refer to $\sum_{p\in P} l(p)f(p)$ as *the cumulative delay* in the later sections.

To help understand the concept of average end-to-end delay, we use the term: "envision" in the above paragraph. In reality, there exists only one stream through each edge (i, j) instead of multiple fractional flows and peer j can reproduce any part of the stream content it receives and send it to other peers. Therefore, the actual data rate on an edge (i, j) is $\max_{t\in R} \sum_{p\in P_{ij}^t} f(p)$, where P_{ij}^t is the set of fractional flows through edge (i, j) to receiver t. Figure 5.1 shows an illustration with example.

Next we provide a formal description of the problem.

5.1.2 The MADPS Problem

Definition 4. ***Minimum Average End-to-End Delay P2P Streaming Problem (MADPS problem):*** *Given the capacity and data rate constraints that are mentioned in this section, the MADPS problem is to devise a streaming scheme which minimizes the maximum average end-to-end streaming delay with all receivers fully served.*

More formally, we formulate the problem in the linear programming framework, as follows:

$$\min \sum_{p\in P} l(p)f(p) \tag{5.2}$$

subject to

$$\sum_{j:(i,j)\in E} \max_{t\in R} \sum_{p\in P_{ij}^t} f(p) \le C_i, \qquad \forall i \in V \tag{5.3}$$

$$\sum_{j:(j,i)\in E} \max_{t\in R} \sum_{p\in P_{ji}^t} f(p) \le I_i, \qquad \forall i \in V \tag{5.4}$$

$$\sum_{p\in P^t} f(p) \ge d_t, \qquad \forall t \in R \tag{5.5}$$

$$f(p) \ge 0, \qquad \forall p \in P. \tag{5.6}$$

Equation (5.3) ensures the sum of actual streaming rates on all edges coming out from the same node i does not exceed the uploading capacity of i. Similarly, Equation (5.4) constrains

the downloading capacity on node i. Equation (5.5) entails each viewer is fully served by the scheduled data rate, where P^t denotes the set of fractional flows to viewer t.

There is no known efficient algorithm with a practically-feasible running time to solve this problem optimally. An exact algorithm for this problem was given in [35] without any analysis for the running time. Simulation results were given only for a very small network (couple of hundreds of nodes and edges). Running time of their algorithm can be prohibitively large for a larger network.

Therefore, we are motivated to develop a near-optimal approximation algorithm with significantly smaller running time.

To ensure a solution exists to the MADPS problem, it is reasonable to assume the total bandwidth resources in P2P networks is sufficient to support the full services on all the viewers. Hence, we deduct the bandwidth requirement in Corollary 4.

Corollary 4. *If the instance of MADPS 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.,*

$$\sum_{i \in V} C_i \geq \sum_{j \in R} d_j. \tag{5.7}$$

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., $\sum_{j \in R} d_j$. Since the flow out of V_{out} cannot exceed its total upload capacities, we have $\sum_{i \in V} C_i \geq \sum_{j \in R} d_j$. The lemma follows. □

In addition, we presume that the download capacity $I_i \geq d_i, \forall i \in V$ for a smooth playback at the receiver.

5.2 Approximation Algorithm

In this section we devise an approximation algorithm to find the near-optimal solution with provable bounds on the worst-case performance and running time.

5.2.1 Overview of Techniques

There are two fundamental techniques used in this work, including *primal-dual schema* and *binary search* based on the result of primal-dual schema.

First, we describe *primal-dual schema* [67, 82]. Given a linear programming problem, also referred to as a *primal problem*, we can convert it to a *dual problem*. We present the mechanics of this conversion in Chapter 3. Detailed methodology can also be found at [82]. Primal and dual problems are in a "mirror" relation. If one problem is a maximization problem, the other problem is a minimization problem, and vise versa. Suppose we have a primal problem: $\max \mathbf{c}^{\mathrm{T}}\mathbf{x}$, and the corresponding dual problem:$\min \mathbf{b}^{\mathrm{T}}\mathbf{y}$. According to the weak duality theorem, if $\mathbf{X}$ and $\mathbf{Y}$ are feasible solutions for the primal and dual problems respectively, it follows that $\mathbf{c}^{\mathrm{T}}\mathbf{X} \leq \mathbf{b}^{\mathrm{T}}\mathbf{Y}$. Moreover, the primal and dual problems share the same optimum, denoted by OPT. Given an approximation factor ρ, ρ bounds $\frac{\mathrm{OPT}}{\mathbf{c}^{\mathrm{T}}\mathbf{x}}$. Since any feasible solution to the dual also provides an upper bound on OPT, the approximation factor can be established by comparing the primal and dual solutions. In light of this, the primal-dual schema starts with a feasible solution for dual problem and relax the conditions for primal problem. Then, it iteratively improves the feasibility of primal conditions and the optimality of the dual solution. the primal-dual schema winds up with feasible solutions for both primal and dual problems. So, the gap between them makes the approximation factor.

In detail, Fastream-II employs the primal-dual schema to solve the delay-bounded maximum streaming rate problem (DBMSR problem) defined as follows.

Definition 5. ***Delay-bounded Maximum Streaming Rate problem (DBMSR problem):*** *Given a bound L on the average delay, i.e., $\sum_{p\in P} l(p)f(p) \leq L$, the DBMSR problem is to devise a streaming scheme which maximizes λ, where $\sum_{p\in P^t} f(p) \geq \lambda d_t, \forall t \in R$.*

In the next step, we can do a binary search on L to finding the smallest λ that satisfies $\lambda \geq 1$. Towards that purpose, a reasonable initial value of L should be set in the range of $[\sum_{j\in R} d_j \cdot \min_{p\in P} l(p), \sum_{j\in R} d_j \cdot \max_{p\in P} l(p)]$. The result of this procedure leads to a near-optimal solution for MADPS problem.

In the rest of this section, we formulate the DBMSR problem by primal-dual schema. Then, we discuss the details of Fastream-II and derive its performance bound.

5.2.2 Formulation about Primal and Dual

We refer to DBMSR problem as the primal problem here, or simply called primal. According to its definition, we formulate the primal as following.

Primal:

$$\max \ \lambda \tag{5.8}$$

subject to

$$\sum_{p\in P_{ij}^t} f(p) \le \sum_{p\in P_{ij}^j} f(p), \qquad \forall (i,j)\in E, \forall t\in R \tag{5.9}$$

$$\sum_{j:(i,j)\in E} \sum_{p\in P_{ij}^j} f(p) \le C_i, \qquad \forall i\in V \tag{5.10}$$

$$\sum_{p\in P^t} f(p) \ge \lambda d_t, \qquad \forall t\in R \tag{5.11}$$

$$\sum_{p\in P} l(p) f(p) \le L \tag{5.12}$$

$$f(p) \ge 0, \qquad \forall p\in P \tag{5.13}$$

$$\lambda \ge 0. \tag{5.14}$$

Since DBMSR problem is an accessory to solving the MADPS problem, its LP expression has close similarity with that of MADPS problem in Section 5.1.2. Equation (5.9) presents the fact that the amount of fractional flow through edge (i,j) to any viewer will always be bounded by the total fractional flow sent to node j, i.e. $\sum_{p\in P_{ij}^j} f(p) = \max_{t\in R} \sum_{p\in P_{ij}^t} f(p)$. Because we attempt to utilize the bandwidth from peers scribing to the standard video quality, it is possible to see the amount of fractional flow to j from all incoming edges of j exceeds viewer j's demand, i.e., $\sum_{i:(i,j)\in E} \sum_{p\in P_{ij}^j} f(p) \ge d_j$. Equation (5.10) ensures no conflicts in terms of the uploading capacities, which actually express the same constraint in Equation (5.4). In terms of the downloading capacities, which can be written as $\sum_{j:(j,i)\in E} \sum_{p\in P_{ji}^i} f(p) \le I_i, \forall i\in V$, we assume $I_i \ge \max_{j\in R} d_j$, which is practical with the wide deployment of high-speed internet. Since the actual flow sent to or relayed by node i cannot be larger than the maximum service demand, expressed by $\max_{j\in R} d_j$, it is reasonable to remove the constraints on the downloading capacities in the LP expression without affecting the optimal solutions. Equation (5.11) means the objective of DBMSR problem is to maximize the minimum demand on nodes. Equation (5.12) puts a bound L on the cumulative delay. As stated in Section 5.2.1, we can conduct a binary search on L until λ is very close to 1 to achieve a solution to the MADPS problem.

Next, we convert the primal to its dual problem, or simply called dual.

Dual:

$$\min \sum_{i\in V} C_i w_i + \varphi L \tag{5.15}$$

subject to

$$\sum_t d_t z_t \ge 1, \qquad \forall t\in R \tag{5.16}$$

$$\sum_{(i,j)\in p, i\neq i'} s_{ij}^t + w_{i'} + \varphi l(p) \geq z_t, \qquad (i',t)\in p, \forall t \in R, \forall p \in P^t \tag{5.17}$$

$$s_{ij}^t \geq 0, \qquad \forall (i,j)\in E, \forall t \in R \tag{5.18}$$

$$w_i \geq 0, \qquad \forall i \in V \tag{5.19}$$

$$z_t \geq 0, \qquad \forall t \in R \tag{5.20}$$

$$\varphi \geq 0, \tag{5.21}$$

where i' is the peer one hop away from the viewer t on routed path.

Generally, there is no direct physical meaning to the dual problem because it comes from a mechanical conversion of the primal problem. To help the analysis on Fastream-II, we hereby assign a logical explanation to the dual after investigating its formulation. We envision each edge (i,j) has multiple copies $(i,j)^1, (i,j)^2, \cdots, (i,j)^{|R|}$, where any copy $(i,j)^t$ exclusively represents to the usage of edge (i,j) for flows to viewer t. Each edge $(i,j)^t$ is associated with a length metric s_{ij}^t, and each node i is associated with a length metric w_i. Thus, we view $\sum_{(i,j)\in p, i\neq i'} s_{ij}^t + w_{i'} + \varphi l(p)$ as the length function associated with flow path p, where φ is the weight associated with the delay metric $l(p)$. According to Equation (5.17), z_t can be comprehended as the shortest length to node t based on the length function.

5.2.3 Approximation Algorithm

APX-Fastream-II is the core part of Fastream-II, which is built with approximation algorithm. APX-Fastream-II proceeds in phases. Each phase is completed by $|R|$ iterations with each iteration satisfying the demand of one viewer. Due to the constraints from LP conditions, each iteration may be completed by multiple steps. Inside each step, we route such amount of fractional flows that can ensure the constraints are not violated. At the end of all phases, APX-Fastream-II will re-scale all the flows to ensure a feasible solution to the primal. We express the k^{th} step in the t^{th} iteration of m^{th} phase by (m,t,k). The initial status is marked by $(0,0,0)$, or simply (0).

We start the algorithm with the following initial settings on length metrics.

$$w_i(0) = \delta/C_i, \qquad \forall i \in V \tag{5.22}$$

$$s_{ij}^t(0) = w_i, \qquad \forall (i,j)\in E, \forall t \in R \tag{5.23}$$

$$\varphi(0) = \delta/L, \tag{5.24}$$

where δ is an input parameter. The proper assignment of it will be discussed in Section 5.2.4.

Throughout the execution of algorithm APX-Fastream-II, it dynamically updates the length metrics, which are used to built the flowing path. Let $w_i(m,t,k), s_{ij}^t(m,t,k), \varphi(m,t,k)$

be the length metrics at the end of step (m,t,k). At step (m,t,k), APX-Fastream-II first computes the shortest path p^* from S to viewer t in terms of the length function $\sum_{(i,j)\in p, i\neq i'} s_{ij}^t(m,t,k-1) + w_{i'}(m,t,k-1) + \varphi(m,t,k-1)l(p)$, where $(i',t) \in p, p \in P^t$. Then, it finds the minimum capacity $C_{\min}$ on nodes along the shortest path, which can be expressed by $C_{\min} = \min_{i\in p^*}\{C_i\}$. Since the previous steps may already route some flows to the viewer, let γ_t be the residual amount of demands unsatisfied on node t, and $x(p) = \min\{\gamma_t, C_{\min}\}$. Next, we route $x(p)/\eta$ amount of flow to t, where $\eta = l(p)x(p)/L$ if $l(p)x(p) > L$; otherwise, $\eta = 1$. So the length bound L and the capacities on the path are not violated in each step. At the end of this step, we update the length metrics as well as the residual demands according to Equations (5.25)-(5.28):

$$w_i(m,t,k) = w_i(m,t,k-1)\cdot\left[1+\epsilon\cdot f(m,t,k)/C_i\right], \quad \forall i \in p^*\setminus\{t\} \tag{5.25}$$

$$s_{ij}^t(m,t,k) = w_i(m,t,k), \forall i \in p^*\setminus\{t\}, \forall (i,j)\in E, \forall t \in R \tag{5.26}$$

$$\varphi(m,t,k) = \varphi(m,t,k-1)\cdot\prod_{j\in p^*\cap R}\left[1+\epsilon\cdot L_j(m,t,k)/L\right], \tag{5.27}$$

$$\gamma_i(m,t,k) = \gamma_i(m,t,k-1) - f(m,t,k), \forall i \in p^*\setminus\{t\} \tag{5.28}$$

where $f(m,t,k)$ is the amount of flow routed in current procedure (m,t,k) and $L_j(m,t,k)$ means the cumulative delay of the routed flow through node j which is on the path p^* at step (m,t,k). Mathematically, it can be expressed by $L_j(m,t,k) = l(p_j^*)x(p_j^*)$, where p_j^* is the segmental path from S to j on path $p*$. We can observe in each step for every capacity-saturated node i on the routing path, all the length metrics regarding i increase by a factor of $1+\epsilon$. Since the assignments of s_{ij}^t are identical in Equation (5.26), we simply use s_i to represent all s_{ij}^t.

We repeat the steps until the demand of viewer t is fully satisfied. Then we call the end of iteration t, and start the iteration for next viewers which has positive residual demand in the current phase. After the last step of a phase, all viewers have no residual demands, i.e., $\gamma_t = 0, \forall t \in R$. Then, we start a new round of phase $m+1$ after resetting the residual demands equal to viewer's actual demands, i.e., $\gamma_t = d_t, \forall t \in R$. The whole procedure completes as soon as $W(m,t,k) \geq 1$. Obviously, the cumulative flows routed in all phases may strongly violate the capacity and average delay constraints. Define $F(p)$ as the cumulative flows routed in all phases through path p. To obtain a feasible solution to the primal problem, we need to scale down each $F(p)$ by a factor of $\log_{1+\epsilon} 1/\delta$. We will justify the correctness of this scaling down factor in Section 5.2.4.

The detailed procedures about the approximation algorithm are presented in Algorithm 5, where the function SHORTEST-PATH$(\cdot)$ represents any feasible shortest path algorithm employed by the user [90–92]. We continue a binary search on L by repeating Algorithm 5 until λ tends to 1, denoted as $\lambda \to 1$. The result of the binary search will provide a near-optimal solution to MADPS problem.

Algorithm 5 APX-Fastream-II($G, \{C_i\}, \{s_i\}, R, \epsilon$): Approximation algorithm for the DBMSR problem

$\varphi = \delta / L$
for all $i \in V$ **do**
  $w_i = \delta / C_i$
  $s_i = \delta / C_i$
end for
for all $p \in P$ **do**
  $f(p) = 0$
  $F(p) = 0$
end for
while $W < 1$ **do**
  for all $t \in R$ **do**
    $\gamma_t = d_t$
    while $W < 1$ AND $\gamma_t > 0$ **do**
      $p = \text{SHORTEST-PATH}(S, t, \{s_i^t + \varphi l_{ij}\})$
      $C_{\min} = \min_{i \in p}\{C_i\}$
      $x(p) = \min\{\gamma_t, C_{\min}\}$
      $L(p) = l(p)x(p)$
      if $L(p) > L$ **then**
        $f(p) = x(p) \cdot L / L(p)$
        $L(p) = L$
      else
        $f(p) = x(p)$
      end if
      $\gamma_t = \gamma_t - x(p)$
      for all $i \in p \setminus \{t\}$ **do**
        $w_i = w_i \cdot [1 + \epsilon \cdot f(p)/C_i]$
        $s_i = w_i$
        $p' = p$
        repeat
          $f(p') = f(p)$
          $\varphi = \varphi(1 + \epsilon \cdot l(p')f(p')/L)$
          $F(p') = F(p') + f(p')$
          $p' = p' \setminus \{v\}$, where v is the target node on path p'
        until $p' = \{S\}$
        $\gamma_i = \gamma_i - f(m, t, k)$
      end for
    end while
  end for
end while
for all $p \in P$ **do**
  $F(p) = F(p) / \log_{1+\epsilon} \frac{1}{\delta}$
end for
$\lambda = \min_{t \in R} \frac{\sum_{p \in P^t} F(p)}{d_t}$

5.2.4 Algorithm Analysis

In this section, we formally analyze the algorithm and prove the approximation factor. To facilitate the analysis, we make some definitions. Let $W = \sum_{i \in V} C_i w_i + \varphi L$ be the metric minimized by the dual. Let ζ_t be the shortest length from S to t, i.e.,

$$\zeta_t = \min_{p \in P^t} \sum_{(i,j) \in p, i \neq i'} s_{ij}^t + w_{i'} + \varphi l(p). \tag{5.29}$$

Here ζ_t actually represents and interprets the meaning of z_t. Besides, we define

$$\alpha = \sum_t \Big(d_t \zeta_t \Big). \tag{5.30}$$

Lemma 7. *Denote the optimal solution to the dual by OPT(W). When OPT(W) is obtained, α is 1.*

Proof. We prove this lemma by contradiction. As we know, α represents $\sum_t d_t z_t$ in the dual. Let $W = W'$ when $\alpha = 1$. For the sake of contradiction, we assume $W' > \mathrm{OPT}(W)$, where $\mathrm{OPT}(W)$ is achieved when $\alpha = \alpha^* > 1$. Then, we scale down α^* to 1. Towards that, we can divide all the s_{ij}^t and φ by a factor of $\sum_t d_t z_t$. As a result, w_i will proportionally scale down the same factor. Consequently, it leads to an update on W with a new value W', where $W' = \mathrm{OPT}(W) / \sum_t d_t z_t$. According to the assumption, W' should be larger than $\mathrm{OPT}(W)$. However, because $\sum_t d_t z_t > 1$, we have $W' = \mathrm{OPT}(W) / \sum_t d_t z_t < \mathrm{OPT}(W)$, which contradicts the assumption. Thus, the lemma follows. □

Define β as the minimum value of W/α, i.e., $\beta = \min W/\alpha$. We conclude the following theorem.

Theorem 6. *The optimal solution to the dual, denoted as OPT(W), is equivalently to the optimal solution β under the same constraints in the dual.*

Proof. From the definition of β,we know that $\beta = \min W/\alpha$. Suppose β is achieved when $\alpha = \alpha^* > 1$. We can always proportionally scale down all the s_{ij}^t and φ by multiplying a factor of $1/\alpha^*$. As a result, $\alpha = 1$. Since W will scale down with the same factor, W/α will keep the optimal value β. That is to say we can always find the optimal solution β with $\alpha = 1$.

According to Lemma 7, it follows that $\alpha = 1$ when $\mathrm{OPT}(W)$ is achieved. Therefore, we can conclude the problem of finding $\mathrm{OPT}(W)$ for the dual is equivalently to solving the optimization problem for W/α. This completes the proof. □

In Algorithm 5, we update the length metrics s_i, w_i, φ on the routing path. In terms of that, we can conclude the following.

Lemma 8. *w_i increases at least by a factor of $1+\epsilon$ for every C_i units of flow through node $i, \forall i \in V$.*

Proof. Denote the flows routed through node i in every step of Algorithm 5 by $f_1^i, f_2^i, \cdots, f_N^i$, respectively, where N represents the total number of flows through node i at the end phase m. Besides, we denote $w_i(k)$ as the updated value after flow f_k^i is routed through node i, where $1 \leq k \leq N$.

Let $w_i(0)$ be the initial value of w_i and $w_i(m)$ be the value of w_i at the end phase m. According to Algorithm 5, we know $f_k^i \leq C_i, \forall i, \forall 1 \leq k \leq N$. Therefore, upon completing the algorithm, we have

$$\begin{aligned} w_i(m) &= w_i(0) \cdot \prod_{k=1}^{N} (1 + \epsilon \cdot f_k^i / C_i) \\ &\geq w_i(0) \cdot \prod_{k=1}^{N} (1+\epsilon)^{f_k^i / C_i} \\ &= w_i(0) \cdot (1+\epsilon)^{\Sigma_{k=1}^{N} f_k^i / C_i}. \end{aligned}$$

Consequently, we can observe

$$\log_{1+\epsilon} \frac{w_i(m)}{w_i(0)} \geq \sum_{k=1}^{N} f_k^i / C_i.$$

This completes the proof. □

Based on proof idea of Lemma 8, we can easily deduce the following corollaries.

Corollary 5. *s_i increases at least by a factor of $1+\epsilon$ for every C_i units of flow through node $i, \forall i \in V$.*

Corollary 6.

$$\log_{1+\epsilon} \frac{\varphi(m)}{\varphi(0)} \geq \sum_{p \in P} l(p) f(p) / L,$$

where $f(p)$ represents the cumulative amount of flows through path p at the end of phase m.

Given the assumption that the total bandwidth resources in P2P networks is sufficient to support the full services of all the viewers, we can do a binary search on L so as to find the smallest λ that satisfies $\lambda \geq 1$. According to the weak-duality theorem, it follows that $\beta \geq \lambda \geq 1$.

Lemma 9. *Given $\beta \geq 1$, we have*

$$\beta \leq \frac{\epsilon(M-1)}{(1-\epsilon)\ln\frac{1-\epsilon}{(|V|+1)\delta}}.$$

Proof. We start the proof by analyzing the change on W on each step. At the end of this analysis, we will carry out the cumulative increment on W when algorithm stops.

Let $p(m,t,k)$ be the shortest path found at procedure (m,t,k), and $f(m,t,k)$ be the quantity of flow routed through path $p(m,t,k)$. Because in our algorithm we assign $s_i = w_i$ for any procedure (m,t,k), we can simplify the length function as

$$\sum_{(i,j)\in p, i\neq i'} s_i^t + w_{i'} + \varphi l(p) = \sum_{(i,j)\in p}(w_i + \varphi l_{ij}), \tag{5.31}$$

where $(i',t) \in p$. Consequently, we can carry out the following.

Since the objective is to find the cumulative increment, we can think of the change on length metrics w_i and φ regarding node i at procedure (m,t,k), where $i \neq t$, will hold until procedure $(m,i,0)$ without loss on the final cumulative increment on W.

$$\begin{aligned}
&W(m,t,k) - W(m,t,k-1)\\
=&C_{i'} \cdot \Big(w_{i'}(m,t,k) - w_{i'}(m,t,k-1)\Big) +\\
&+ \Big(\varphi(m,t,k) - \varphi(m,t,k-1)\Big)\cdot L\\
\leq &\sum_{i\in p(m,t,k)\setminus\{t\}} \Big(C_i \cdot w_i(m,t,k-1)\epsilon f(m,t,k)/C_i\Big) +\\
&+ \Big(\varphi(m,t,k-1)\epsilon L(m,t,k)/L\Big)\cdot L\\
=&\epsilon \cdot \Big[\sum_{i\in p(m,t,k)\setminus\{t\}} \Big(w_i(m,t,k-1)f(m,t,k)\Big) +\\
&+ \varphi(m,t,k-1)L(m,t,k)\Big].
\end{aligned}$$

Let K_{mt} be the number of steps in a given iteration t of phase m, $\zeta_t(m,t,k)$ be the shortest path at the end of procedure (m,t,k), and $l(m,t,k)$ be the cumulative latency on path $p(m,t,k)$. We have

$$W(m,t+1,0) - W(m,t,0)$$

$$
\begin{aligned}
\leq & \epsilon \cdot \sum_{k=1}^{K_{mt}} \Big[\sum_{i \in p(m,t,k) \setminus \{t\}} \Big(w_i(m,t,k-1) f(m,t,k) \Big) + \\
& + \varphi(m,t,k-1) L(m,t,k) \Big] \\
= & \epsilon \cdot \sum_{k=1}^{K_{mt}} \Big[f(m,t,k) \cdot \sum_{i \in p(m,t,k) \setminus \{t\}} \Big(w_i(m,t,k-1) \Big) + \\
& + \varphi(m,t,k-1) l(m,t,k) \Big] \\
= & \epsilon \cdot \sum_{k=1}^{K_{mt}} \Big[f(m,t,k) \cdot \sum_{(i,j) \in p(m,t,k)} \Big(w_i(m,t,k-1) + \\
& + \varphi(m,t,k-1) l_{ij} \Big) \Big] \\
= & \epsilon \cdot \sum_{k=1}^{K_{mt}} f(m,t,k) \cdot \zeta_t(m,t,k-1) \\
\leq & \epsilon \cdot d_t \zeta_t(m,t,k).
\end{aligned}
$$

For brevity on notations, we define $W(m)$ as the value of W at the end of phase m, and make a similar definition for $\alpha(m)$. Then, it follows that

$$
\begin{aligned}
& W(m) - W(m-1) \\
= & W(m, |R|, K_{m|R|}) - W(m,0,0) \\
\leq & \epsilon \cdot \sum_{t=1}^{|R|} \Big(d_t \zeta_t(m,t,K_{m|R|}) \Big) \\
\leq & \epsilon \alpha(m).
\end{aligned} \tag{5.32}
$$

Combining the property of $W(m)/\alpha(m) \geq \beta$ with Equation (5.32), we can carry out

$$W(m) \leq \frac{W(m-1)}{1-\epsilon/\beta}.$$

In light of the initial settings, $w_i(0) = \delta/C_i$ and $\varphi(0) = \delta/L$. Thus, we obtain $W(0) = (|V|+1)\delta$.

Given $m \geq 1$ and $\beta \geq 1$, it follows that

$$W(m) \leq \frac{(|V|+1)\delta}{(1-\epsilon/\beta)^m}$$

$$
\begin{aligned}
&= \frac{(|V|+1)\delta}{1-\epsilon/\beta}(1+\frac{\epsilon}{\beta-\epsilon})^{m-1} \\
&\leq \frac{(|V|+1)\delta}{1-\epsilon/\beta} e^{\frac{\epsilon(m-1)}{\beta-\epsilon}} \\
&\leq \frac{(|V|+1)\delta}{1-\epsilon} e^{\frac{\epsilon(m-1)}{(1-\epsilon)\beta}}.
\end{aligned}
$$

Let the last phase in the algorithm be numbered by M. It follows that $1 \leq W(M) \leq \frac{(|V|+1)\delta}{1-\epsilon} e^{\frac{\epsilon(M-1)}{(1-\epsilon)\beta}}$. Hence, we carry out

$$\beta \leq \frac{\epsilon(M-1)}{(1-\epsilon)\ln\frac{1-\epsilon}{(|V|+1)\delta}}.$$

Thus, the lemma follows. □

Lemma 10. *Algorithm 5 generates a feasible streaming solution that makes* $\lambda \geq \frac{M-1}{\log_{1+\epsilon} 1/\delta}$.

Proof. At the end of the $(M-1)^{\text{th}}$ phase, $W(M-1) \leq 1$ for all node i. Thus, we deduct $s_i(M-1) = w_i(M-1) \leq 1/C_i$.

From Lemma 8 and Corollary 5, we know w_i and s_i increase at least by a factor of $1+\epsilon$ for every C_i units of flow through node i. Denoting the total flow through node i as F_i, we can carry out

$$
\begin{aligned}
F_i &\leq C_i \log_{1+\epsilon} \frac{w_i(M-1)}{w_i(0)} \\
&\leq C_i \log_{1+\epsilon} \frac{1/C_i}{\delta/C_i} \\
&= C_i \log_{1+\epsilon} \frac{1}{\delta}.
\end{aligned}
$$

Therefore, dividing all the flows through node i by a scaling factor of $\log_{1+\epsilon}\frac{1}{\delta}$, we obtain feasible flows through i without violating its uploading capacity C_i.

Applying the scaling factor, we can get feasible flows received by t of a total value $(M-1)d_t/\log_{1+\epsilon}\frac{1}{\delta}$ units. Accordingly, a feasible λ will follow

$$\lambda \geq \frac{(M-1)d_t/\log_{1+\epsilon}\frac{1}{\delta}}{d_t}$$

$$= \frac{(M-1)}{\log_{1+\epsilon} \frac{1}{\delta}}.$$

□

Theorem 7. *The result of Algorithm 5 follows the property of* $\sum_{p \in P} l(p)f(p) \leq L$.

Proof. According to Corollary 6, in our procedure every time we route every flow with a cumulative delay of L, we increase φ by at least a factor of $1+\epsilon$.

Because $W(M-1) < 1$, we deduce that $\varphi(M-1) < 1/L$. Thus, in the first $M-1$ phases, the cumulative delay is at most $L \cdot \log_{1+\epsilon} \frac{\varphi(M-1)}{\varphi(0)} = L \cdot \log_{1+\epsilon} \frac{1}{\delta}$, i.e., $\sum_{p \in P} l(p)f(p) \leq L \cdot \log_{1+\epsilon} \frac{1}{\delta}$.

In the final procedure of the algorithm, we scale down all the flows proportionally by a scaling factor. Thus, applying the scaling factor of $\log_{1+\epsilon} \frac{1}{\delta}$, we have

$$\begin{aligned} \sum_{p \in P} l(p)f(p) &\leq \frac{L \log_{1+\epsilon} \frac{1}{\delta}}{\log_{1+\epsilon} \frac{1}{\delta}} \\ &= L. \end{aligned}$$

The theorem follows. □

Theorem 8. *The approximation factor, denoted as* ρ, *is* $1+\omega$.

Proof. From Lemma 10, we have a feasible solution $\lambda = \frac{M-1}{\log_{1+\epsilon} \frac{1}{\delta}}$. It follows that

$$\begin{aligned} \frac{\beta}{\lambda} &= \frac{\beta \log_{1+\epsilon} \frac{1}{\delta}}{(M-1)} \\ &= \frac{\epsilon \ln \frac{1}{\delta}}{(1-\epsilon)\ln(1+\epsilon) \ln \frac{1-\epsilon}{(|V|+1)\delta}}. \end{aligned}$$

Let $\delta = \left(\frac{1-\epsilon}{|V|+1}\right)^{1/\epsilon}$. We have

$$\begin{aligned} \frac{\beta}{\lambda} &\leq \frac{\epsilon \ln \frac{1}{\delta}}{(1-\epsilon)\ln(1+\epsilon) \ln \frac{1-\epsilon}{(|V|+1)\delta}} \\ &= \frac{\epsilon}{(1-\epsilon)^2 \ln(1+\epsilon)} \\ &\leq \frac{\epsilon}{(1-\epsilon)^2(\epsilon - \epsilon^2/2)} \end{aligned}$$

$$\leq (1-\epsilon)^{-3}.$$

According to the strong duality theorem, if the dual has the optimal solution β, the primal also has an optimal value, denoted as $\mathrm{OPT}(\lambda)$, such that $\mathrm{OPT}(\lambda) = \beta$. Therefore, the approximation factor ρ can be obtained by

$$\begin{aligned}\rho &= \max \frac{\mathrm{OPT}(\lambda)}{\lambda} \\ &= \max \frac{\beta}{\lambda}.\end{aligned}$$

Now, we make an assignment of $\omega = (1-\epsilon)^{-3} - 1$. We have $\rho = 1 + \omega$.

Thus, the proof is complete.

□

5.2.5 Running Time

In this section, we analyze the bound on running time. We define maximum binary search bound on L as $\Gamma = \sum_{j \in R} d_j \cdot \max_{p \in P} l(p)$.

Theorem 9. *Suppose the shortest path algorithm employed will consume a running time of Ψ. The running time of Fastream-II is $O(\epsilon^{-2}\Psi|V|\log|V|\log\Gamma)$.*

Proof. According to weak duality theorem, we have $\frac{\beta}{\lambda} \geq 1$, which deduces

$$\frac{\beta}{M-1}\log_{1+\epsilon}\frac{1}{\delta} > 1.$$

So the number of phases $M < 1 + \beta \log_{1+\epsilon}\frac{1}{\delta}$. Because $\delta = \left(\frac{1-\epsilon}{|V|+1}\right)^{1/\epsilon}$, it follows that

$$M = \lceil \frac{\beta}{\epsilon}\log_{1+\epsilon}\frac{|V|+1}{1-\epsilon} \rceil$$

If Algorithm 5 does not stop within $2\lceil \frac{1}{\epsilon}\log_{1+\epsilon}\frac{|V|+1}{1-\epsilon}\rceil$ phases, we must have $\beta \geq 2$. We know $\mathrm{OPT}(\lambda) = \beta$ and we are pursuing $\mathrm{OPT}(\lambda) = 1$. In the case of $\beta \geq 2$, we break the current call for Algorithm 5, and continue the binary search on L. So each call for Algorithm 5 will have $2\lceil \frac{1}{\epsilon}\log_{1+\epsilon}\frac{|V|+1}{1-\epsilon}\rceil = O(\epsilon^{-2}\log|V|)$ phases.

In order to compute the total running time, we need to calculate the number of steps in each call for Algorithm 5. It is easy to see at every step except the the last step in an iteration, we increase either w_i of some node or φ by a factor at least $1+\epsilon$. So the number of steps exceeds the number of iterations by at most

$$|V| \log_{1+\epsilon} \frac{w_i(M-1)}{w_i(0)} = |V| \log_{1+\epsilon} \frac{1}{\delta} = O(\epsilon^{-2}|V| \log |V|). \tag{5.33}$$

Also, the maximum number of iterations in all phases is $|R| \cdot O(\epsilon^{-2} \log |V|) = O(\epsilon^{-2}|R| \log |V|)$. Combining this with Equation (5.33), we have the total number of steps in each call for Algorithm 5 is $O(\epsilon^{-2}(|V| + |R|) \log |V|) = O(\epsilon^{-2}|V| \log |V|)$.

Considering the number of calls for Algorithm 5 in binary search is bounded by $\log \Gamma$. Consequently, we can carry out the running time of Fastream-II is bounded by $O(\epsilon^{-2}\Psi|V| \log |V| \log \Gamma)$. The theorem follows. □

5.2.6 Fastream-II-D

We briefly discuss the possibility of practical deployment of Fastream-II-D here. Fastream-II-D can be initiated by servers or superpeers in the network by broadcasting the initial value of ϵ. Any existing distributed shortest path algorithm can be employed by Fastream-II-D. In each step, peers will update the length metric locally and send back updated φ to server. Server will synchronize the parameters on each peers in the network and monitor the whole procedures of Fastream-II-D until the minimum latency is found. Considering network dynamics and startup delay, we can make Fastream-II-D a hybrid of Fastream-II and heuristics. On node arrival or departure, heuristics will be called to help peers influenced by network churn. Once the average delay is higher than the preset threshold, Fastream-II will be called. Additionally, strategy such as backup links can be used by Fastream-II-D to make it adaptive to the network churn.

5.3 Simulation Study

In this section we report our simulation results. We implement Fastream-II and conduct simulations of a P2P video streaming system with up to 1000 peers. We simulate a live streaming session of diverse streaming qualities from one single source with 10 Mbps upload capacity. To compare with Fastream-I, we assume an identical streaming rate of 350 Kbps demanded by all peers. Existing studies show that the network bandwidth exhibits rich diversity [21, 81]. Based on this, we set 30%, 50%, 15% and 5% peers have an upload capacity of 200 Kbps, 1 Mbps, 2 Mbps and 10Mbps, respectively. We assume there are

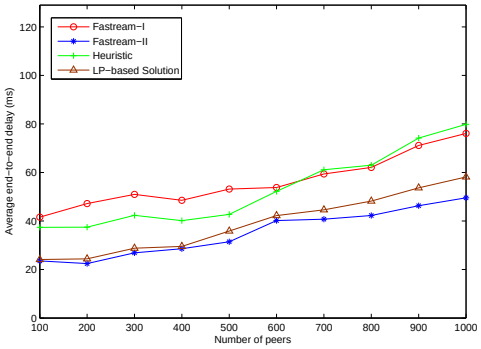


Figure 5.2: Average end-to-end latency

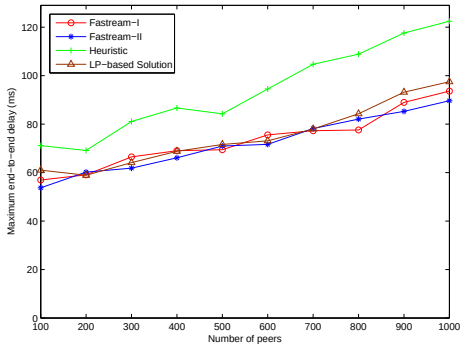


Figure 5.3: Maximum end-to-end latency

no constraints on downloading capacities in the simulation because download capacities are generally larger than the maximum demanded streaming rate. In this simulation study, we compare our algorithm with three other algorithms: Fastream-I, a heuristic approach [21] and a LP-based approach [35]. The heuristic in [21] 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 [21]. The LP-based approach in [35] applies several linear programming techniques to obtain an optimal average delay, such as Lagrangian relaxation and subgradient algorithm. Please note, to reduce computational costs, [35] restricts the potential connections for each peer. This actually lowers 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 paper.

In this simulation, we judge the performance based on two metrics, including *average end-to-end delay* and *maximum end-to-end delay*. The average end-to-end delay is defined as the weighted average latency from the source to all receivers. From Figure 5.2, we can see the average delays occurring on different algorithms when the number of peers increases. In general, the average delays increase with augmentation of the number of peers on all

the algorithms. Compared with the other three algorithms, Fastream-II achieves the lowest average delay. It is noticeable that LP-based solution has a slightly longer average delay than Fastream-II. Because LP-based solution restricts the connections for each peer, it can not reflect the optimal average delay. It is also interesting to observe that the heuristic approach exhibits lower average delay than Fastream-I when the network size is relatively small (under 600 nodes). When the network size approaches 700 nodes, the heuristic has an average latency that is above that of Fastream-I. Moreover, Fastream-I cannot compete with Fastream-II and LP-based approaches when the objective is to minimize the average delay. This may result from when Fastream-I attempts to minimize the maximum delay, it sacrifices the average delay. From the plot, we also see the average latency from the heuristic has a faster growth rate than other three approaches when the network size increases. Such phenomena implies that the heuristic method is not scalable in terms of delay performance for large networks.

We also measure the maximum delay and show it in Figure 5.3, which is defined as the worst-case end-to-end delay observed in the simulation experiments. It is apparent that the worst-case performance of Fastream-I outperforms the heuristic. It is reasonable since the objective of Fastream-I is to minimize the maximum delay. Furthermore, it is interesting to see Fastream-II and LP-based solutions also present excellent performance in terms of the maximum delay, although they are designed to minimize the average delay instead of the maximum delay. Several reasons may exist for this: (1) the bound on Fastream-I is $O(\sqrt{\log n})$, which does not provide the solution that is close enough to the optimal solution so as to significantly outperform Fastream-II and LP-based solutions; (2) minimizing average delay has no big influence on the maximum delay so Fastream-II and LP-based solutions also perform well in the measurement of the maximum delay.

Conclusively, our simulation results illustrate the excellent effectiveness of Fastream-II in terms of two metrics: average delay and maximum delay.

5.4 Summary

We present the design of Fastream-II and derive a near-optimal approximation bound for its core component APX-Fastream-II. To achieve a tractable theoretical analysis, we assume no network dynamics in the first stage of algorithm design. Although the assumption is strong in practical P2P applications, the value of this work lies in the theoretical framework and analysis, which sheds light on the practical design. To reduce the complexity of the problem, we focus only on minimizing the communication delay. For packet scheduling, there exists a vast array of solutions. The mesh built from our algorithm can adopt any of these scheduling algorithms to yield low-delay streaming.

Chapter 6

Minimizing Multi-channel P2P Streaming Delay

In this chapter, we describe the problem of minimizing source-to-port delay in multi-channel P2P streaming, i.e. MSPDS problem. First, we formulate MSPDS problem in LP. We devise Fastream-III to solve MSPDS problem based on primal-dual schema. Then, we prove the near-optimum performance assurance of Fastream-III.

6.1 Problem Formulation

In this section, we present the system model and define the minimum average source-to-port delay P2P streaming (MSPDS) problem.

6.1.1 System Model

We denote the set of channels by H, the set of source nodes by S, the set of ports by T, and the set of internal servants as Υ, where $\Upsilon = V \setminus T$ (V is the complete set of nodes in the network). We assume each channel $\alpha \in H$ streams from a single source node $S_\alpha \in S$ to a set of ports $T_\alpha \subset T$. Ports may receive the data from the source node directly or indirectly from internal servants. We assume ports of each channel demand the complete data content of that channel in order to serve the viewers. Suppose port j is serving channel α that has a constant streaming rate of r_α units/second. We denote f_{ij} as the rate at which peer i streams to peer j. If port j receives the aggregated non-identical streams at r_α units/second from its parents, we call port j as *fully served* [21]. Mathematically, the fully served requirement of port j can be expressed as $\sum_{i:i\in\Lambda_j} f_{ij} = r_\alpha$, where Λ_j is the set of parents of peer j. A fully served port can deliver any part of the streaming content to the subscribed viewers.

We call the stream from the source s to one port j as the P2P unicast flow to j. Each P2P unicast flow U_{sj} may consist of streams from multiple P2P paths, called *fractional flows* [35]. Each fractional flow $p \in U_{sj}$ has the arrival latency $l(p)$ from the source to port, i.e., *source-to-port delay*, where $l(p) = \sum_{(i,j)\in p} l_{ij}$. We define the average source-to-port delay of the unicast flow U_{sj} as the weighted average of source-to-port latencies of all its fractional flows, where the weight is the portion of fractional flow rate to the total streaming rate. Denote $f(p)$ as the streaming rate of fractional flow p. For a unicast flow U_{sj}, the weighted average of source-to-port latencies can be expressed by

$$\frac{1}{r_\alpha} \sum_{p\in U_{sj}} l(p)f(p).$$

To stream multiple channel contents from their sources to their respective ports, we can envision multiple unicast flows from the sources to ports. Thus, the *average source-to-port delay in multi-channel P2P streaming* is defined as the weighted average latency of all fractional flows from all sources to their respective ports, which can be described by

$$\frac{1}{\sum_{\alpha\in H} |T_\alpha| r_\alpha} \sum_{p\in P^T} l(p)f(p), \tag{6.1}$$

where P^T is the set of streaming paths to all the ports. Since the term $\sum_{\alpha\in H} |T_\alpha| r_\alpha$ does not affect the optimal solution, i.e., the solution that minimizes (6.1) also minimizes $\sum_{p\in P^T} l(p)f(p)$, we will focus on minimizing $\sum_{p\in P^T} l(p)f(p)$. For ease of presentation, we simply refer to $\sum_{p\in P^T} l(p)f(p)$ as *the cumulative delay* in the later sections.

To help understand the concept of average source-to-port delay, we use the term: "envision" in the above paragraph. In reality, there exists only one stream through each edge (i, j) instead of multiple fractional flows and peer j can reproduce any part of the stream content it receives and send it to other peers. Therefore, the actual data rate on an edge (i, j) is $\sum_\alpha \max_{t\in T_\alpha} \sum_{p\in P_{ij}^{\alpha t}} f(p)$, where $P_{ij}^{\alpha t}$ is the set of fractional flows of channel α through edge (i, j) to receiver t.

Next we provide a formal description of the MSPDS problem.

6.1.2 The MSPDS Problem

For a tractable problem space, we assume that the ports have been selected for each channel. Our objective in this problem is to assign peers to each channel as the internal servants so as to minimize the average source-to-port streaming delay.

Definition 6. ***Minimum Source-to-Port Delay P2P Streaming Problem (MSPDS problem):*** *Given the capacity and data rate constraints that are mentioned in Section 6.1.1,*

the MSPDS problem is to devise a streaming scheme which minimizes the average source-to-port streaming delay with all ports fully served.

More formally, we formulate the problem in the linear programming framework, as follows:

$$\min \sum_{p \in P^T} l(p) f(p). \tag{6.2}$$

There is no known efficient algorithm with a practically-feasible running time to optimally solve this problem. Therefore, we're motivated to develop a near-optimal algorithm with bounded running time.

To ensure a solution exists to the MSPDS problem, it is reasonable to assume the total bandwidth resources in P2P networks is sufficient to support the streaming requirement to all the ports. Hence, we deduct the bandwidth requirement in Corollary 7.

Corollary 7. *If the instance of MSPDS problem has a solution, the sum of the upload capacities of internal servants, must be no less than the sum of fully served streaming rates at all ports, i.e.,*

$$\sum_{i \in \Upsilon} C_i \geq \sum_{\alpha \in H} |T_\alpha| r_\alpha. \tag{6.3}$$

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 at least the flow into T, i.e., $\sum_{\alpha \in H} |T_\alpha| r_\alpha$. Since the flow out of V_{out} cannot exceed its total upload capacities, we have $\sum_{i \in \Upsilon} C_i \geq \sum_{\alpha \in H} |T_\alpha| r_\alpha$. The lemma follows. □

In addition, we presume that the download capacity $I_i \geq r_i, \forall i \in V$ for a smooth playback at the receiver.

6.2 Methodology about Fastream-III

In this section we present the methodology of Fastream-III. We will emphasize the details about APX-Fastream-III–the core component of Fastream-III, an approximation algorithm with provable bounds on the worst-case performance and running time.

6.2.1 Overview of Techniques

For a feasible solution, we start with the assumption of a static network—i.e., no churn on network nodes and links. In this way, we can devise a framework that is analytically achievable. Accordingly, the method will be most suitable for the scenario where a service provider deploys a set-top box at viewers' homes. In that case, even when a viewer turns off the TV, the set-top box can still contribute its bandwidth to other viewers.

To find the near-optimal solution for MSPDS problem, Fastream-III takes two steps. First, it employs the primal-dual schema to solve the delay-bounded maximum streaming rate problem (DBMPSR problem) described in Definition 7.

Definition 7. ***Delay-bounded Maximum Port Streaming Rate problem (DBMPSR problem):*** *Given a bound L on the cumulative delay, i.e., $\sum_{p\in P^T} l(p)f(p) \leq L$, the DBMPSR problem is to devise a streaming scheme which maximizes μ, where $\sum_{p\in P^{\alpha t}} f(p) \geq \mu r_\alpha, \forall t \in T_\alpha, \forall \alpha \in H$.*

We allow one port to serve in multiple serving swarms. Thus, we define $P^{\alpha t}$ as the set of paths from S_α to port t.

In the next step, we can do a binary search on L to find the smallest μ that satisfies $\mu \geq 1$. Towards that purpose, a reasonable initial value of L should be set in the range of $[\sum_{\alpha\in H} |T_\alpha| r_\alpha \cdot \min_{p\in P^T} l(p), \sum_{\alpha\in H} |T_\alpha| r_\alpha \cdot \max_{p\in P^T} l(p)]$. The result of this procedure leads to a near-optimal solution for MSPDS problem.

The fundamental technique used in first step for DBMPSR problem is *primal-dual schema* [93]. Given a linear programming problem, also referred to as a *primal problem*, we can convert it to a *dual problem*. Primal and dual problems are in a "mirror" relation. If one problem is a maximization problem, the other problem is a minimization problem, and vice versa. Suppose we have a primal problem: $\max \mathbf{c}^{\mathrm{T}}\mathbf{x}$, and the corresponding dual problem:$\min \mathbf{b}^{\mathrm{T}}\mathbf{y}$. According to the weak duality theorem, if $\mathbf{X}$ and $\mathbf{Y}$ are feasible solutions for the primal and dual problems respectively, it follows that $\mathbf{c}^{\mathrm{T}}\mathbf{X} \leq \mathbf{b}^{\mathrm{T}}\mathbf{Y}$. Moreover, the primal and dual problems share the same optimum, denoted by OPT. Given an approximation factor ρ, ρ bounds $\frac{\mathrm{OPT}}{\mathbf{c}^{\mathrm{T}}\mathbf{x}}$. Since any feasible solution to the dual also provides an upper bound on OPT, the approximation factor can be established by comparing the primal and dual solutions. In light of this, the primal-dual schema starts with a feasible solution for dual problem and relax the conditions for primal problem. Then, the algorithm iteratively improves the feasibility of primal conditions and the optimality of the dual solution. Our algorithm winds up with feasible solutions for both primal and dual problems. So, the gap between them makes the approximation factor.

In the rest of this section, we formulate the DBMPSR problem by primal-dual schema. Then, we discuss the details of APX-Fastream-III and derive its performance bound.

6.2.2 Formulation about Primal and Dual

We formulate the DBMPSR problem in the linear programming (LP) framework. Here, we refer to DBMPSR problem as the primal problem, or simply called primal.

Primal:

$$\max \ \mu \tag{6.4}$$

subject to

$$\sum_{p \in P_{ij}^{\alpha t}} f(p) \leq \sum_{p \in P_{ij}^{\alpha j}} f(p), \forall (i,j) \in E, \forall t \in T_\alpha, \forall \alpha \in H \tag{6.5}$$

$$\sum_{\alpha \in H} \sum_{j:(i,j) \in E} \sum_{p \in P_{ij}^{\alpha j}} f(p) \leq C_i, \qquad \forall i \in \Upsilon \tag{6.6}$$

$$\sum_{p \in P^{\alpha t}} f(p) \geq \mu r_\alpha, \qquad \forall t \in T_\alpha, \forall \alpha \in H \tag{6.7}$$

$$\sum_{p \in P^T} l(p) f(p) \leq L. \tag{6.8}$$

$$f(p) \geq 0, \qquad \forall p \in P \tag{6.9}$$

$$\mu_t \geq 0, \qquad \forall t \in T. \tag{6.10}$$

Equation (6.5) presents the fact that the amount of fractional flow through edge (i, j) to any port t from source S_α will always be bounded by the total fractional flow sent to node j from the same source, where j could be either internal servant or port in the serving swarm of channel α. Equation (6.6) ensures no conflicts in the uploading capacities. In terms of the downloading capacities, which can be written as $\sum_{\alpha \in H} \sum_{j:(j,i) \in E} \sum_{p \in P_{ji}^{\alpha i}} f(p) \leq I_i, \forall i \in V$, because the actual flow sent to or relayed by node i cannot be larger than its demand, it is reasonable to remove the constraints on the downloading capacities in the LP expression without affecting the optimal solutions. Equation (6.7) means the objective of DBMPSR problem is to maximize the minimum demand on ports. Equation (6.8) puts a bound L on the cumulative delay.

Next, we convert the primal to its dual problem, or simply called dual.

Dual:

$$\min \sum_{i \in \Upsilon} C_i w_i + L\varphi \tag{6.11}$$

subject to

$$\sum_{\alpha \in H} \sum_{t \in T_\alpha} z_{\alpha t} r_\alpha \geq 1, \qquad \forall t \in T_\alpha, \forall \alpha \in H \tag{6.12}$$

$$\sum_{(i,j)\in p, i\neq i'} s_{ij}^{\alpha t} + w_{i'} + \varphi l(p) \geq z_{\alpha t}, \qquad (i',t)\in p, \forall p \in P^{\alpha t}, \forall t \in T_\alpha, \forall \alpha \in H \tag{6.13}$$

$$w_i + \varphi l(p) \geq |T| \sum_{(i,j)\in p} s_{ij}^{\alpha j}, \qquad \forall \alpha \in H, \forall p \in P \setminus P^{\alpha t}. \tag{6.14}$$

$$s_{ij}^{\alpha k} \geq 0, \qquad \forall (i,j) \in E, \forall k \in V \setminus S \tag{6.15}$$

$$w_i \geq 0, \qquad \forall i \in V \tag{6.16}$$

$$z_{\alpha t} \geq 0, \qquad \forall t \in T \tag{6.17}$$

$$y_t \geq 0, \qquad \forall t \in T \tag{6.18}$$

$$\varphi \geq 0. \tag{6.19}$$

where i' is the peer one hop away from the port t on routed path, and $P^{\alpha t}$ is the set of paths from S_α to t.

Generally, there is no direct physical meaning to the dual problem because it comes from a mechanical conversion of the primal problem. To help the analysis on Fastream-III, we hereby assign a logical explanation to the dual after investigating its formulation. We envision each edge (i,j) has multiple copies, where any copy $(i,j)^{\alpha t}$ exclusively represents to the usage of edge (i,j) for flows from S_α to port t. Each edge $(i,j)^{\alpha t}$ is associated with a length metric $s_{ij}^{\alpha t}$, and each node i is associated with a length metric w_i. Thus, we view $\sum_{(i,j)\in p, i\neq i'} s_{ij}^{\alpha t} + w_{i'} + \varphi l(p)$ as the length function associated with flow path p, where φ is the weight associated with the delay metric $l(p)$. According to Equation (6.13), $z_{\alpha t}$ can be comprehended as the shortest length from S_α to node t based on the length function.

6.2.3 Approximation Algorithm

APX-Fastream-III proceeds in phases. Each phase is completed by $|H|$ iterations with each iteration satisfying, the demands of all ports for one channel. Due to the constraints expressed in the LP conditions, each iteration may be completed in multiple steps. Inside each step, we route such amount of fractional flows that can ensure the constraints are not violated. At the end of all phases, APX-Fastream-III will re-scale all the flows to ensure a feasible solution to the primal. We express the k^{th} step in the α^{th} iteration of m^{th} phase by (m,α,k). The initial status is marked by $(0,0,0)$, or simply (0).

We start the algorithm with the following initial settings on length metrics.

$$w_i(0) = \delta/C_i, \qquad \forall i \in V \tag{6.20}$$

$$s_{ij}^{\alpha t}(0) = w_i, \qquad \forall (i,j) \in E, \forall t \in T_\alpha \forall \alpha \in H \tag{6.21}$$

$$\varphi(0) = \delta/L, \tag{6.22}$$

where δ is an input parameter, H is the set of channels, T_α is the set of ports of channel α, and $\alpha \in H$. The proper assignment of it will be discussed in Section 6.2.4.

Throughout the execution of algorithm APX-Fastream-III, it dynamically updates the length metrics, which are used to build the flowing path. Let $w_i(m,\alpha,k), s_{ij}^{\alpha t}(m,\alpha,k), \varphi(m,\alpha,k)$ be the length metrics at the end of step (m,α,k). At step (m,α,k), APX-Fastream-III first computes the shortest path tree from S_α to all ports in T_α, who have positive remaining demands. Comparing with APX-Fastream-II, this procedure makes significant improvement in that we do not need to call shortest path search algorithm for each port. Instead, we call one the shortest path algorithm one time for all ports in the channel. At this step, the length function used in constructing the shortest path tree is $\sum_{(i,j)\in p, i\neq i'} s_{ij}^{\alpha t}(m,\alpha,k-1) + w_{i'}(m,\alpha,k-1) + \varphi(m,\alpha,k-1)l(p)$, where $(i',t) \in p$. Then, it finds the minimum capacity $C_{\min}^{\alpha t}$ on nodes along each shortest path to its destination t, which can be expressed by $C_{\min}^{\alpha t} = \min_{i\in p^{\alpha t}}\{C_i\}$ where $p^{\alpha t}$ is the shortest path. Since the previous steps may already route some flows to the port, let $\gamma_{\alpha t}$ be the remaining demand on port t regarding channel α, and $x(p^{\alpha t}) = \min\{\gamma_{\alpha t}, C_{\min}^{\alpha t}\}$. Next, we route $x(p^{\alpha t})/\eta$ amount of flow to t, where $\eta = \sum_{t\in T_\alpha} l(p^{\alpha t})x(p^{\alpha t})/L$ if $l(p^{\alpha t})x(p^{\alpha t}) > L$; otherwise, $\eta = 1$. So the length bound L and the capacities on the path are not violated in each step. At the end of this step, we update the length metrics as well as the residual demands according to Equations (6.23)-(6.26).

$$w_i(m,\alpha,k) = w_i(m,\alpha,k-1)\cdot\left[1+\epsilon\cdot f_i(m,\alpha,k)/C_i\right], \quad \forall i \in V\setminus T \tag{6.23}$$

$$s_{ij}^{\alpha t}(m,\alpha,k) = w_i(m,\alpha,k), \forall (i,j)\in E, \forall t\in T_\alpha \tag{6.24}$$

$$\varphi(m,\alpha,k) = \varphi(m,\alpha,k-1)\cdot\left[1+\epsilon\cdot L(m,\alpha,k)/L\right], \tag{6.25}$$

$$\gamma_{\alpha t}(m,\alpha,k) = \gamma_{\alpha t}(m,\alpha,k-1) - f_t(m,\alpha,k), \forall t\in T_\alpha, \tag{6.26}$$

where $f_i(m,t,k)$ is the amount of flow routed through i in current procedure (m,t,k) and $L(m,\alpha,k)$ means the cumulative delay of all routed flows through node at step (m,t,k), i.e., $\sum_{t\in T_\alpha} l(p^{\alpha t})x(p^{\alpha t})$. We can observe in each step for every capacity-saturated node i on the routing path, all the length metrics regarding i increase by a factor of $1+\epsilon$. Since the assignments of $s_{ij}^{\alpha t}$ are identical with w_i in Equation (6.24), we simply use s_i to represent all $s_{ij}^{\alpha t}$.

We repeat the steps until the demand of all ports in T_α is fully satisfied. Then we call the end of iteration α, and start the iteration for next channel $\alpha+1$ in the current phase. After the last step of a phase, all ports have no residual demands, i.e., $\gamma_{\alpha t} = 0, \forall t\in T_\alpha$. Then, we start a new round of phase $m+1$ after resetting the residual demands equal to port's actual demands, i.e., $\gamma_{\alpha t} = r_\alpha, \forall t\in T_\alpha, \forall\alpha$. Let $W = \sum_{i\in V} C_i w_i + L\varphi$ be the metric minimized by the dual. The whole procedure completes as soon as $W(m,\alpha,k) \geq 1$. Obviously, the cumulative flows routed in all phases may strongly violate the capacity and average delay constraints. Define $F(p)$ as the cumulative flows routed in all phases through path p. To

Algorithm 6 APX-Fastream-III$(G, H, S, \{T_\alpha\}, \{C_i\}, \{r_\alpha\}, \delta, \epsilon)$: Approximation algorithm for the DBMPSR problem

$\varphi = \delta / L$
for all $i \in V$ **do**
  $s_i = \delta / C_i$
end for
for all $p \in P^T$ **do**
  $f(p) = 0$
  $F(p) = 0$
end for
while $W < 1$ **do**
  for all $\alpha \in H$ **do**
    $\gamma_{\alpha t} = r_\alpha$
    while $W < 1$ AND $\gamma_{\alpha t} > 0$ **do**
      $P'_\alpha = \text{SHORTEST-PATH-TREE}(S_\alpha, T_\alpha, \{w_i + \varphi l_{ij}\})$
      $L(\alpha) = \sum_{p \in P'_\alpha} l(p)x(p)$
      for all $p \in P'_\alpha$ **do**
        $C^{\alpha t}_{\min} = \min_{i \in p}\{C_i\}$
        $x(p) = \min\{\gamma_{\alpha t}, C^{\alpha t}_{\min}\}$
        if $L(\alpha) > L$ **then**
          $f(p) = x(p) \cdot L / L(p)$
        else
          $f(p) = x(p)$
        end if
        $F(p) = F(p) + f(p)$
        $\gamma_{\alpha t} = \gamma_{\alpha t} - f(p)$
      end for
      for all $i \in P'_\alpha \setminus T_\alpha$ **do**
        $f_i = \sum_{p: i \in p \& p \in P'_\alpha} \{f(p)\}$
        $w_i = w_i \cdot [1 + \epsilon \cdot f_i(p) / C_i]$
        $s_i = w_i$
        $\varphi = \varphi(1 + \epsilon \cdot L(\alpha) / L)$
      end for
    end while
  end for
end while
for all $p \in P^T$ **do**
  $F(p) = F(p) / \log_{1+\epsilon} \frac{1}{\delta}$
end for
$\mu = \min_{t \in T_\alpha, \alpha \in H} \frac{\sum_{p \in P^{\alpha t}} F(p)}{r_\alpha}$

obtain a feasible solution to the primal problem, we need to scale down each $F(p)$ by a factor of $\log_{1+\epsilon} 1/\delta$. We will justify the correctness of this scaling down factor in Section 6.2.4.

The detailed procedures about the approximation algorithm are presented in Algorithm 6, where the function SHORTEST-PATH-TREE(A, B, C) builds the shortest path tree from A to all ports in B using length metric C and returns the set of paths in the shortest path tree. It can be any feasible shortest path algorithm employed by the user [92]. We continue a binary search on L by repeating Algorithm 6 until μ tends to 1, denoted as $\mu \to 1$. The result of the binary search will provide a near-optimal solution to MADPS problem.

6.2.4 Algorithm Analysis

In this section, we are going to analyze the algorithm and justify the performance of Fastream-III. First, we make some definitions. Let $\zeta_{\alpha t}$ be the shortest length from S_α to t, i.e.,

$$\zeta_{\alpha t} = \min_{p \in P^t} \sum_{(i,j) \in p, i \neq i'} s_{ij}^{\alpha t} + w_{i'} + \varphi l(p), \tag{6.27}$$

where $t \in T_\alpha$. Comparing $\zeta_{\alpha t}$ with Equation (6.13), we can observe $\zeta_{\alpha t}$ actually stands for $z_{\alpha t}$.

Next, we define

$$\theta = \sum_{\alpha \in H} \sum_{t \in T_\alpha} \Big(r_\alpha \zeta_{\alpha t} \Big). \tag{6.28}$$

Let OPT(W) be the optimal solution to the dual.

Lemma 11. *When OPT(W) is obtained, $\theta = 1$.*

Proof. We use contradiction in the proof. We know that θ represents $\sum_{\alpha \in H} \sum_{t \in T_\alpha} z_{\alpha t} r_\alpha$ in the dual. Let $W = W'$ when $\theta = 1$. For the sake of contradiction, we assume $W' >$ OPT(W), where OPT(W) is achieved when $\theta = \theta^* > 1$. Then, we scale down θ^* to 1. Towards that, we can divide all the $s_{ij}^{\alpha t}$, w_i and φ by a factor of $\sum_{\alpha \in H} \sum_{t \in T_\alpha} z_{\alpha t} r_\alpha$. As a result, W will proportionally scale down the same factor with a new value W', where $W' =$ OPT$(W)/\sum_{\alpha \in H} \sum_{t \in T_\alpha} z_{\alpha t} r_\alpha$. According to the assumption, W' should be larger than OPT(W). However, because $\sum_{\alpha \in H} \sum_{t \in T_\alpha} z_{\alpha t} r_\alpha > 1$, we have $W' =$ OPT$(W)/\sum_{\alpha \in H} \sum_{t \in T_\alpha} z_{\alpha t} r_\alpha <$ OPT(W), which contradicts the assumption.

Thus, the lemma follows. □

Next, we denote the minimum value of W/θ as β, i.e., $\beta = \min W/\theta$. We denote the optimal solution to the dual as OPT(W). Then, we can claim the following theorem.

Theorem 10. *OPT(W) is equivalently to the optimal solution β.*

Proof. From the definition of β,we know that $\beta = \min W/\theta$. Suppose β is achieved when $\theta = \theta^* > 1$. We can always proportionally scale down all the $s_{ij}^{\alpha t}$, w_i and φ by multiplying a factor of $1/\theta^*$. As a result, $\theta = 1$. Because W will scale down with the same factor, W/θ will keep the optimal value β. That is to say we can always find the optimal solution β with $\theta = 1$.

According to Lemma 11, we find that $\theta = 1$ when OPT(W) is achieved. Therefore, we can conclude the problem of finding OPT(W) for the dual is equivalently to solving the optimization problem for W/θ. This completes the proof. □

In Algorithm 6, we update the length metrics s_i, w_i, φ on the routing path. In those terms, we can conclude the following.

Lemma 12.

$$\log_{1+\epsilon} \frac{w_i(m)}{w_i(0)} \geq \sum_{k=1}^{N} f_k^i / C_i.$$

Proof. First, we denote the flows routed through node i in every step of Algorithm 6 by $f_1^i, f_2^i, \cdots, f_N^i$, respectively, where N represents the total number of flows through node i at the end phase m. Besides, we denote $w_i(k)$ as the updated value after flow f_k^i is routed through node i, where $1 \leq k \leq N$.

Let $w_i(0)$ be the initial value of w_i and $w_i(m)$ be the value of w_i at the end phase m. According to Algorithm 6, we know $f_k^i \leq C_i, \forall i, \forall 1 \leq k \leq N$. Therefore, upon completing the algorithm, we have

$$\begin{aligned} w_i(m) &= w_i(0) \cdot \prod_{k=1}^{N} (1 + \epsilon \cdot f_k^i / C_i) \\ &\geq w_i(0) \cdot \prod_{k=1}^{N} (1+\epsilon)^{f_k^i / C_i} \\ &= w_i(0) \cdot (1+\epsilon)^{\Sigma_{k=1}^{N} f_k^i / C_i}. \end{aligned}$$

Consequently, we can observe

$$\log_{1+\epsilon} \frac{w_i(m)}{w_i(0)} \geq \sum_{k=1}^{N} f_k^i / C_i.$$

This completes the proof. □

Similar to the proof of Lemma 12, we can deduce the following lemmas.

Lemma 13. *s_i increases at least by a factor of $1+\epsilon$ for every C_i units of flow through node $i, \forall i \in V$.*

Lemma 14.

$$\log_{1+\epsilon} \frac{\varphi(m)}{\varphi(0)} \geq \sum_{p \in P} l(p) f(p)/L,$$

where $f(p)$ represents the cumulative amount of flows through path p at the end of phase m.

Suppose that the total bandwidth resources in P2P networks is sufficient to support the demands on all ports, we then do a binary search on L to find the smallest μ that satisfies $\mu \geq 1$. According to the weak-duality theorem, it follows that $\beta \geq \mu \geq 1$. Thus, we can conclude the following.

Lemma 15.

$$\beta \leq \frac{\epsilon(M-1)}{(1-\epsilon)\ln \frac{1-\epsilon}{(n+1)\delta}}.$$

Proof. First, let us analyze the change on W on each step. Then, at the end of this analysis, we will carry out the cumulative increment on W when algorithms stops.

We denote $P'_\alpha(m,\alpha,k)$ as the shortest path tree found at procedure (m,α,k), and $f_i(m,\alpha,k)$ be the quantity of flow routed through node i at step (m,α,k). Because we make such assignment that $s_i = w_i$ for any procedure (m,t,k), we can simplify the length function as

$$\sum_{(i,j)\in p, i\neq i'} s_i^t + w_{i'} + \varphi l(p) = \sum_{(i,j)\in p} (w_i + \varphi l_{ij}), \tag{6.29}$$

where $(i',t) \in p$. Consequently, we can carry out the following.

$$\begin{aligned}
& W(m,\alpha,k) - W(m,\alpha,k-1) \\
=& \sum_{i \in P'_\alpha(m,\alpha,k)} C_i \cdot \Big(w_i(m,\alpha,k) - w_i(m,\alpha,k-1) \Big) + \\
& + \Big(\varphi(m,\alpha,k) - \varphi(m,\alpha,k-1) \Big) \cdot L \\
=& \sum_{i \in P'_\alpha(m,\alpha,k)} \Big(C_i \cdot w_i(m,\alpha,k-1) \epsilon f_i(m,\alpha,k)/C_i \Big) +
\end{aligned}$$

$$
\begin{aligned}
&+\Big(\varphi(m,\alpha,k-1)\epsilon L(m,\alpha,k)/L\Big)\cdot L\\
=&\epsilon\cdot\Big[\sum_{i\in P'_\alpha(m,\alpha,k)}\Big(w_i(m,\alpha,k-1)f_i(m,\alpha,k)\Big)+\\
&+\varphi(m,\alpha,k-1)L(m,\alpha,k)\Big]\\
=&\epsilon\cdot\Big[\sum_{i\in P'_\alpha(m,\alpha,k)}\Big(w_i(m,\alpha,k-1)f_i(m,\alpha,k)\Big)+\\
&+\sum_{t\in T_\alpha}\varphi(m,\alpha,k-1)l_t(m,\alpha,k)f_t(m,\alpha,k)\Big]\\
\le&\epsilon\cdot\Big[\sum_{t\in T_\alpha}\sum_{i\in p^{\alpha t}(m,\alpha,k)}\Big(w_i(m,\alpha,k-1)f_i(m,\alpha,k)\Big)+\\
&+\sum_{t\in T_\alpha}\varphi(m,\alpha,k-1)l_t(m,\alpha,k)f_t(m,\alpha,k)\Big]\\
=&\epsilon\sum_{t\in T_\alpha}\Big[f_t(m,\alpha,k)\cdot\Big(\sum_{i\in p^{\alpha t}(m,\alpha,k)}w_i(m,\alpha,k-1)+\\
&+\varphi(m,\alpha,k-1)l_t(m,\alpha,k)\Big)\Big],
\end{aligned}
$$

where $p^{\alpha t}(m,\alpha,k)$ is the shortest path to viewer t for channel α at step (m,α,k).

Let $K_{m\alpha}$ be the number of steps in a given iteration α of phase m, $\zeta_{\alpha t}(m,\alpha,k)$ be the shortest path at the end of procedure (m,α,k), and $l_{\alpha t}(m,\alpha,k)$ be the length on path from S_α to t at step (m,α,k). We have

$$
\begin{aligned}
&W(m,\alpha+1,0)-W(m,\alpha,0)\\
=&W(m,\alpha,K_{mt})-W(m,\alpha,0)\\
\le&\epsilon\cdot\sum_{k=1}^{K_{m\alpha}}\sum_{t\in T_\alpha}\Big[f_t(m,\alpha,k)\cdot\Big(\sum_{i\in p^{\alpha t}(m,\alpha,k)}w_i(m,\alpha,k-1)+\\
&+\varphi(m,\alpha,k-1)l_t(m,\alpha,k)\Big)\Big]\\
\le&\epsilon\cdot\sum_{k=1}^{K_{m\alpha}}\sum_{t\in T_\alpha}f_t(m,\alpha,k)\cdot\zeta_{\alpha t}(m,\alpha,k-1)\\
=&\epsilon\cdot r_\alpha\zeta_{\alpha t}(m,\alpha,k).
\end{aligned}
$$

For brevity on notations, we define $W(m)$ as the value of W at the end of phase m, and make a similar definition for $\theta(m)$. Then, it follows that

$$\begin{aligned} & W(m) - W(m-1) \\ =& W(m, |H|, K_{m|H|}) - W(m, 0, 0) \\ \leq& \epsilon \cdot \sum_{\alpha=1}^{|H|} \Big(r_\alpha \zeta_{\alpha t}(m, \alpha, K_{m|H|}) \Big) \\ \leq& \epsilon\theta(m). \end{aligned} \tag{6.30}$$

Combining the property of $W(m)/\theta(m) \geq \beta$ with Equation (6.30), we can carry out

$$W(m) \leq \frac{W(m-1)}{1-\epsilon/\beta}.$$

In light of the initial settings, $w_i(0) = \delta/C_i$ and $\varphi(0) = \delta/L$. Thus, we obtain $W(0) = (n+1)\delta$, where $n = |\Upsilon|$.

Given $m \geq 1$ and $\beta \geq 1$, it follows that

$$\begin{aligned} W(m) &\leq \frac{(n+1)\delta}{(1-\epsilon/\beta)^m} \\ &= \frac{(n+1)\delta}{1-\epsilon/\beta}(1 + \frac{\epsilon}{\beta-\epsilon})^{m-1} \\ &\leq \frac{(n+1)\delta}{1-\epsilon/\beta} e^{\frac{\epsilon(m-1)}{\beta-\epsilon}} \\ &\leq \frac{(n+1)\delta}{1-\epsilon} e^{\frac{\epsilon(m-1)}{(1-\epsilon)\beta}}. \end{aligned}$$

Let the last phase in the algorithm be numbered by M. It follows that $1 \leq W(M) \leq \frac{(n+1)\delta}{1-\epsilon} e^{\frac{\epsilon(M-1)}{(1-\epsilon)\beta}}$. Hence, we carry out

$$\beta \leq \frac{\epsilon(M-1)}{(1-\epsilon)\ln\frac{1-\epsilon}{(n+1)\delta}}.$$

Thus, the lemma follows. □

Lemma 16. *APX-Fastream produces a feasible streaming solution that has the property of* $\mu > \frac{M-1}{\log_{1+\epsilon} 1/\delta}$.

Proof. It is simple to see that $W(M-1) \leq 1$ for all node i at the end of the $(M-1)^{\text{th}}$ phase. Based on that, we can obtain $s_i(M-1) = w_i(M-1) \leq 1/C_i$.

From Lemma 12 and Corollary 13, we know w_i and s_i increase at least by a factor of $1+\epsilon$ for every C_i units of flow through node i. Denoting the total flow through node i as F_i, we can carry out

$$\begin{aligned} F_i &\leq C_i \log_{1+\epsilon} \frac{w_i(M-1)}{w_i(0)} \\ &\leq C_i \log_{1+\epsilon} \frac{1/C_i}{\delta/C_i} \\ &= C_i \log_{1+\epsilon} \frac{1}{\delta}. \end{aligned}$$

In light of that, dividing all the flows through node i by a scaling factor of $\log_{1+\epsilon} \frac{1}{\delta}$, we obtain feasible flows through i without violating its uploading capacity C_i.

Applying the scaling factor, we can get feasible flows received by t of a total value $(M-1)r_\alpha / \log_{1+\epsilon} \frac{1}{\delta}$ units. Accordingly, a feasible μ will follow

$$\begin{aligned} \mu &> \frac{(M-1)r_\alpha / \log_{1+\epsilon} \frac{1}{\delta}}{r_\alpha} \\ &= \frac{(M-1)}{\log_{1+\epsilon} \frac{1}{\delta}}. \end{aligned}$$

□

Theorem 11. *Algorithm 6 generates a result with the property of* $\sum_{p \in P^T} l(p)f(p) \leq L$.

Proof. According to Corollary 14, of our procedure, every time we route every flow with a cumulative delay of L, we increase φ by at least a factor of $1+\epsilon$.

Because $W(M-1) < 1$, we deduct that $\varphi(M-1) < 1/L$. Thus, in the first $M-1$ phases, the cumulative delay is at most $L \cdot \log_{1+\epsilon} \frac{\varphi(M-1)}{\varphi(0)} = L \cdot \log_{1+\epsilon} \frac{1}{\delta}$, i.e., $\sum_{p \in P^T} l(p)f(p) \leq L \cdot \log_{1+\epsilon} \frac{1}{\delta}$.

In the final procedure of the algorithm, we scale down all the flows proportionally by a scaling factor. Thus, applying the scaling factor of $\log_{1+\epsilon} \frac{1}{\delta}$, we have

$$\sum_{p \in P^T} l(p)f(p) \leq \frac{L \log_{1+\epsilon} \frac{1}{\delta}}{\log_{1+\epsilon} \frac{1}{\delta}}$$

$$= L.$$

The theorem follows. □

Theorem 12. *The approximation factor, denoted as ρ, is $1 + \omega$.*

Proof. From Lemma 16, we have a feasible solution $\mu = \frac{M-1}{\log_{1+\epsilon} \frac{1}{\delta}}$. It follows that

$$\begin{aligned}\frac{\beta}{\mu} &= \frac{\beta \log_{1+\epsilon} \frac{1}{\delta}}{(M-1)} \\ &= \frac{\epsilon \ln \frac{1}{\delta}}{(1-\epsilon)\ln(1+\epsilon)\ln \frac{1-\epsilon}{(n+1)\delta}}.\end{aligned}$$

Let $\delta = \left(\frac{1-\epsilon}{n+1}\right)^{1/\epsilon}$. We have

$$\begin{aligned}\frac{\beta}{\mu} &\leq \frac{\epsilon \ln \frac{1}{\delta}}{(1-\epsilon)\ln(1+\epsilon)\ln \frac{1-\epsilon}{(n+1)\delta}} \\ &= \frac{\epsilon}{(1-\epsilon)^2 \ln(1+\epsilon)} \\ &\leq \frac{\epsilon}{(1-\epsilon)^2(\epsilon - \epsilon^2/2)} \\ &\leq (1-\epsilon)^{-3}.\end{aligned}$$

According to the strong duality theorem, if the dual has the optimal solution β, the primal also has an optimal value, denoted as $\mathrm{OPT}(\mu)$, such that $\mathrm{OPT}(\mu) = \beta$. Therefore, the approximation factor ρ can be obtained by

$$\begin{aligned}\rho &= \max \frac{\mathrm{OPT}(\mu)}{\mu} \\ &= \max \frac{\beta}{\mu}.\end{aligned}$$

Now, we make an assignment of $\omega = (1-\epsilon)^{-3} - 1$. We have $\rho = 1 + \omega$.

Thus, the proof is complete.

□

6.2.5 Running Time

In this section, we work on deducing the bound on running time. First, we define maximum binary search bound on L as $\Gamma = \sum_{\alpha \in H} |T_\alpha| r_\alpha \cdot \max_{p \in P^T} l(p)$. We denote the running time of the shortest path algorithm is Ψ.

Theorem 13. *The running time of Fastream-III is* $O(\epsilon^{-2} \Psi n \log n \log \Gamma)$.

Proof. According to weak duality theorem, we have $\frac{\beta}{\mu} \geq 1$, which deduces

$$\frac{\beta}{M-1} \log_{1+\epsilon} \frac{1}{\delta} > 1.$$

So the number of phases $M < 1 + \beta \log_{1+\epsilon} \frac{1}{\delta}$. Because $\delta = \left(\frac{1-\epsilon}{n+1}\right)^{1/\epsilon}$, it follows that

$$M = \lceil \frac{\beta}{\epsilon} \log_{1+\epsilon} \frac{n+1}{1-\epsilon} \rceil$$

If Algorithm 6 does not stop within $2\lceil \frac{1}{\epsilon} \log_{1+\epsilon} \frac{n+1}{1-\epsilon} \rceil$ phases, we must have $\beta \geq 2$. We know $\mathrm{OPT}(\mu) = \beta$ and we're pursuing $\mathrm{OPT}(\mu) = 1$. In the case of $\beta \geq 2$, we break the current call for Algorithm 6, and continue the binary search on L. So each call for Algorithm 6 will have $2\lceil \frac{1}{\epsilon} \log_{1+\epsilon} \frac{n+1}{1-\epsilon} \rceil = O(\epsilon^{-2} \log n)$ phases.

In order to compute the total running time, we need to calculate the number of steps in each call for Algorithm 6. It is easy to see at every step, except the the last step in an iteration, we increase either w_i of some node or φ by a factor at least $1 + \epsilon$. So the number of steps exceeds the number of iterations by at most

$$n \log_{1+\epsilon} \frac{w_i(M-1)}{w_i(0)} = n \log_{1+\epsilon} \frac{1}{\delta} = O(\epsilon^{-2} n \log n). \tag{6.31}$$

Also, the total number of iterations in all phases is bounded by $|H| \cdot O(\epsilon^{-2} \log n) = O(\epsilon^{-2} |H| \log n)$. Combining this with Equation (6.31), we have the total number of steps in each call for Algorithm 6 is $O(\epsilon^{-2}(n + |H|) \log n) = O(\epsilon^{-2} n \log n)$ supposing each channel as at least one port.

Considering the number of calls for Algorithm 6 in binary search is bounded by $\log \Gamma$. Consequently, we can carry out the running time of Fastream-III is bounded by $O(\epsilon^{-2} \Psi n \log n \log \Gamma)$. The theorem follows. □

6.3 Summary

We devise Fastream-III for MSPDS problem and derive a near-optimal approximation bound for its key component APX-Fastream-III. Current result is the first ever solution that solves MSPDS problem. To achieve a tractable theoretical analysis, we assume no network dynamics in the first stage of algorithm design. Although the assumption is strong in practical P2P applications, the value of this work lies in the theoretical framework and analysis, which sheds light on the practical design. To reduce the complexity of the problem, we focus only on minimizing the communication delay. Future work can consider packet scheduling based on the design of Fastream-III.

Chapter 7

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–NAP, 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.

7.1 NAP: An Agent-Based P2P Scheme

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, 94, 95], and channel churn comes from channel switching [13, 96]. In this section, we first formulate the problems of reducing delays caused by the two types of churns: node churn and channel churn, and then we describe the methodology of NAP to reduce these delays.

7.1.1 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 $\mathbf{C}$. Given a channel $C_i \in \mathbf{C}$, 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 $C_i \in \mathbf{C}$ is considered as a streaming session, originating from a single source node S_i to a set of receivers R_i, where $V_i = \{S_i\} \cup R_i$. 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

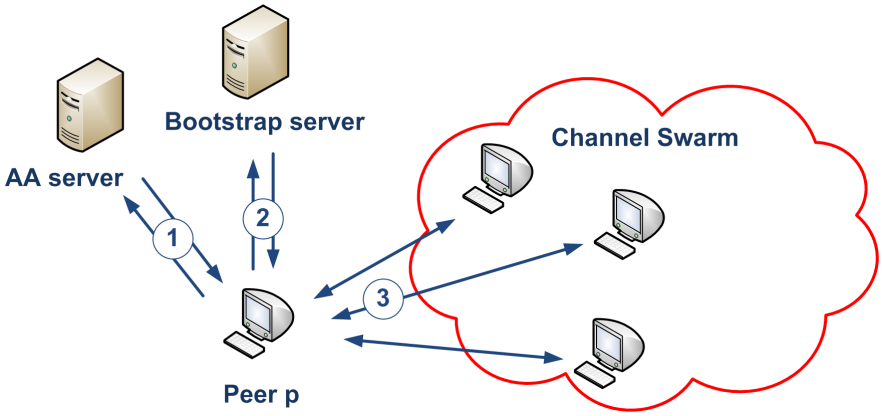


Figure 7.1: General Scheme: (1) *Authentication and authorization*; (2) *Peer list download*; (3) *Contact other peers, retrieve chunk map and more lists of connection candidates, and then schedule downloading plan.*

peer p as *fully served* [21]. We assume that a fully served peer can smoothly play back the streaming content at its original rate of s_i units/second [21].

To understand the switching delay problem, we first describe the general channel switching scheme without switching delay optimization [41]. As illustrated in Figure 7.1, when switching to a new channel C_i, a peer p initiates a connection with an *AA server* for Authentication and Authorization [41]. Afterwards, it contacts with a *bootstrap server* (BS), which is a server maintaining a partial list L of current peers on the overlay of C_i. BS sends p a list L_R of random peers as the connection candidates for p, where $L_R \subset L$ [41]. To distinguish from the bootstrapping peers which are mentioned later in NAP, we call here the list of peers retrieved from BS as *ordinary peers*. Then p requests streaming from peers in L_R. Active peers in L_R response with their IP addresses, data chunk information and additional lists of connection candidates in the swarm. If the cumulative available bandwidth from responding peers in L_R satisfies the streaming rate and the cumulative chunk is complete, p schedules downloading plan and attempts the streaming process with them.

If the above bootstrap procedure is successful, the overlay mesh for channel C_i is updated, and subsequently p receives 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

$$\overline{d}_{sw} = \overline{d}_A + \overline{d}_{ls} + \overline{d}_p + \overline{d}_{sc}, \tag{7.1}$$

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 8. ***Minimum Channel Switching Delay Problem (MCSD problem):*** *Given the average time required to complete each step in channel switching process, i.e., $\overline{d}_A, \overline{d}_{ls}, \overline{d}_p$, and $\overline{d}_{sc}$ as defined above, the MCSD problem is to devise an overlay switching scheme which minimizes the average channel switching delay with the receiver successfully switching to the new channel.*

Measurement studies have revealed that 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 bootstrap 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 that it will be ready to switch channel. As we know, a typical live streaming application, such as PPLive and UUSee, may accommodate hundreds of channels [97]. 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, NAP, suggests a distributed agent-based bootstrap, where each channel maintains a set of agents responsible for bootstrapping in the channel for external peers. Agents in channel C_i, denoted as $\mathbf{A}_i$, are selected peers with predicated long lifetime, more storage and communication capabilities in the overlay. They collect the bootstrap information about available peers in the channel with periodical update, and then exchange such information with the agents in the other channels. It is worth mentioning that when updating, only changed information is collected and distributed to other agents. As shown in Figure 7.2, on the background of current streaming, peer p, after joining the network, registers at one of the agents $a \in \mathbf{A}_i$ in its current channel C_i and sends a the request for proactive bootstrap to other channels $\bigcup_j C_j$, where $j \neq i$. Agent a buffers and periodically forwards such requests to the agents in the other channels. Those agents in other channels authenticate and authorize p's join request. Once the request is approved, they notify a. Given the bootstrapping information stored at a as $\mathbf{I}_a$, a then sends p a list of peers $\bigcup_j D_p^j$ in the other channels, where $D_p^j \subset \mathbf{I}_a$ and j corresponds to channel C_j. Since then, a updates such bootstrapping information if change happens. Bootstrapping data sent to p contains only a partial set of peers stored in a's list for each channel. A partial set of peers is sufficient as long as the available bandwidth in each channel can fully serve p's streaming. In addition, an agent distills 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, the agents pre-schedule the downloading plan for 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 requests a service to the agent in C_j and simultaneously request streaming from ordinary peers in D_p^j. The agent forwards the service request to the

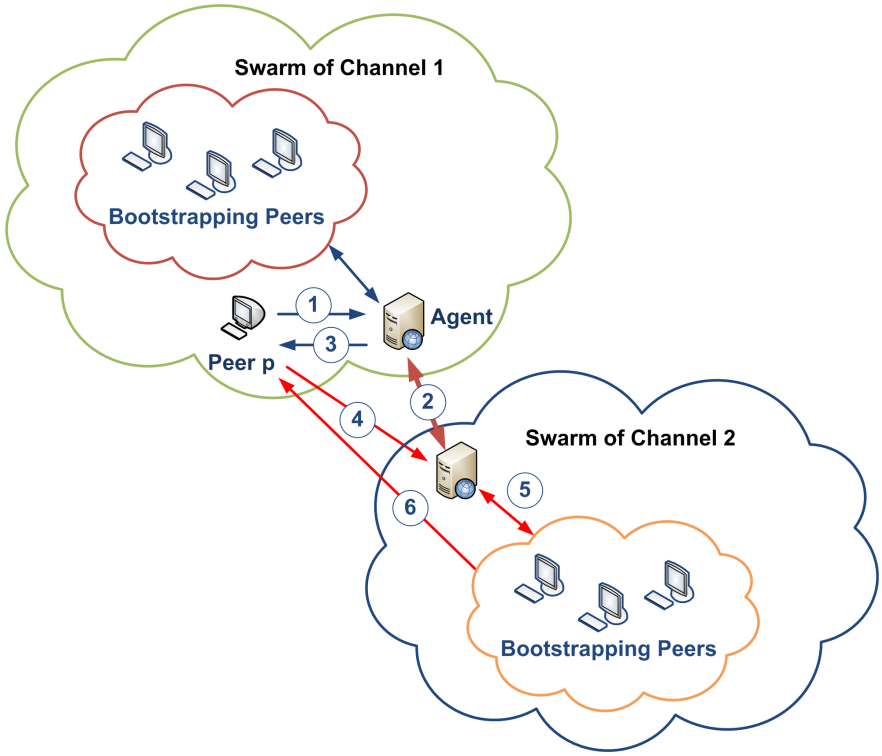
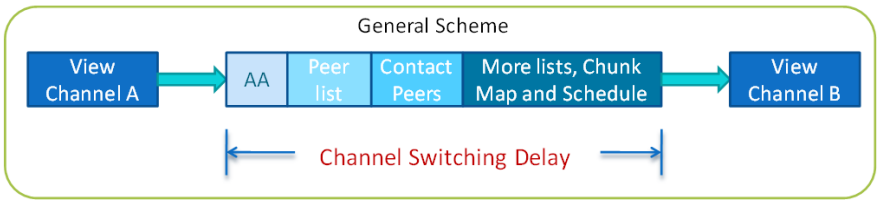


Figure 7.2: NAP: (1) *Peer registers at the agent*; (2) *Agents periodically exchange information with each other, distill bootstrapping peers and pre-schedule the downloading plan based on chunk information on BP*; (3) *Peer downloads peer list and list of agents in other channels*; (4) *When switches channel, peer sends request to agent in that channel and also contacts ordinary peers in the list simultaneously*; (5) *Agent forwards request to bootstrapping peers*; (6) *Bootstrapping peers directly send data to peer.*

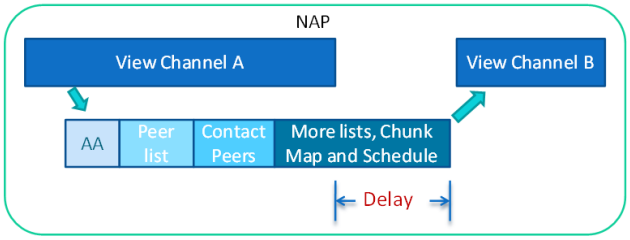
bootstrapping peers in its management for a quick assistance in the initial streaming period. After bootstrapping peers starts streaming, they keep looking for other ordinary peers to take over their streaming job to p. So the agent can recycle back the bootstrapping peers. By way of this proactive bootstrapping, channel switching delays are reduced significantly.

We illustrate the difference between the general scheme (GS) and NAP in terms of channel switching delay in Figure 7.3. Comparatively, NAP parallelizes the threads of viewing behavior and channel switching process (including authentication and authorization (AA), downloading the peer list from BS, contacting the peers in the list, obtaining more peer lists from the contacted peers, building the chunk map, and scheduling the downloading plan) as shown in Figure 7.3b, while GS completes the two threads in sequence as shown in Figure 7.3a. Benefiting from the process parallelization, the channel switching delay in NAP is apparently shorter than that in GS.

To reduce the message overhead for updating data on agents, bootstrapping peers in each



(a) General Scheme



(b) NAP

Figure 7.3: Comparison between General Scheme (GS) and NAP in terms of channel-switching delay.

channel can work only for the initial streaming period T and during that time they handover the streaming to other ordinary peers they found, called *handover peers*. It is reasonable to evaluate T as the expected time that bootstrapping peers need to find handover peers. Because bootstrapping peers can be reused in this way, agents do 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 NAP, where what a peer uploads is independent of what it views [13]. Similar to agent selection, bootstrapping peers should have predicted long lifetime and more bandwidth resources. We further discuss the properties of bootstrapping peers and agents in Section 7.2.

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 only registers at the new agent a but does not send bootstrap request since it has done this previously. Bootstrapping data have time stamps. On p's registration, a compare the time stamp t_a on its bootstrapping data with t_p that is stored in p. If $t_a > t_p$, a updates the bootstrapping data to p.

7.1.2 On Reducing Delay from Peer Churn

In this section, we discuss the protocol to reduce the delay from the other types 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 this dissertation, 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

$$\overline{d}_{re} = \overline{d}_p + \overline{d}_{sc}. \tag{7.2}$$

Comparing with switching delay in Equation (7.1), notice that recovery delay has no $\overline{d}_A$ and $\overline{d}_{ls}$ components since the peer has already been authorized and owns the peer list in current channel. We now define the problem:

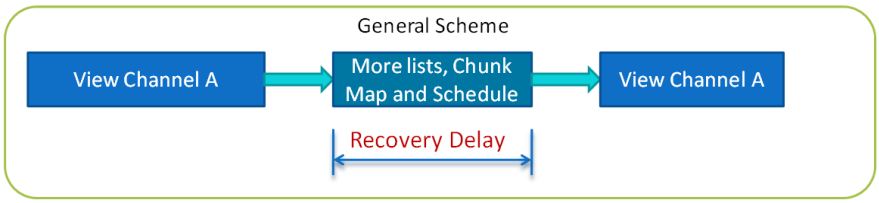
Definition 9. ***Minimum Departure-induced Delay Problem (MDD problem):*** *The MDD problem is to devise an overlay resilience scheme which minimizes the recovery delay during peer departure.*

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 7.1.1. Specifically, given an agent a for peer p that is viewing channel C_i, $\mathbf{I}_a$ stored on a is extended to include its bootstrapping data to the current channel C_i. Likewise, D_p^i is stored and updated in peer p as well.

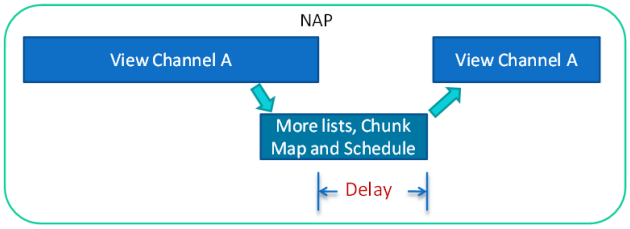
We compare the difference between the general scheme (GS) and NAP in terms of recovery delay in Figure 7.4. Since the viewer continues to watch the same channel when its upstream peer departs, there is no need for it to repeat the steps of contacting AA and BS servers. However, by parallelizing the threads of viewing behavior and re-connection process (including obtaining more peer lists from the contacted peers, building the chunk map, and scheduling the downloading plan), NAP can still save the time in recovering the upstream nodes.

7.1.3 Management of the Agents

The agents exchange information with each other. A resilient way to realize these exchanges is a gossip-based method. There are many existing works in gossip-based schemes, such as [98–100]. Our previous work [98] also presents a gossip-based scheme in P2P scenario. Thus, in this work, we do not focus on how to implement an optimal gossip plan. In addition, an agent may leave the network so peers may not reach the agent when they call for a service to the agent. To alleviate this problem, heart-beat signals are exchanged among agents to timely monitor the existence of the agent. In Section 7.3, we detail the setting of heart-beat signal frequency. Once an agent departure is detected, neighboring agents of the departed agent initiate a process to select a new agent. Neighboring agents exchange the copies of bootstrapping information. So when they select new agent, they put back such information that was stored in the departed agent to the new agent.



(a) General Scheme



(b) NAP

Figure 7.4: Comparison between General Scheme (GS) and NAP in terms of recovery delay.

In terms of the agent selection strategy, there are two major factors in the process: *computational resources* and *predicted lifetime*. Computational resources include the bandwidth, CPU processing speed and storage size, etc. We can pre-set thresholds for such metrics when distilling the agent out of general peers. Predicted lifetime means the estimated time span that the agent candidate can stay in the streaming network. The predication method has close similarity with the super-peer selection process described in [42, 84], where Cox Regression Model and Maximum Likelihood Estimation are employed. Detailed methodology about them can be found in [84]. We will not extend our discussion in that respect.

7.2 Modeling and Analysis

To achieve a tractable solution, we formally model and analyze NAP with queueing theory. In contrast to the traditional Client/Server paradigm and the general P2P file sharing application, the high churn rate and live streaming restriction impose more challenge in modeling a P2P streaming scheme. In the following, we develop a queueing model for NAP considering the failure and restore processes of agents and bootstrapping peers. Although this queueing model is developed for NAP, our study indicates it can also be generally applied to other preemptive schemes after simple modification.

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 14. *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. The sum of two independent Poisson variables also follows Poisson distribution. More precisely, if $X_1 \sim \text{Poisson}(\lambda_1)$ and $X_2 \sim \text{Poisson}(\lambda_2)$, then $(X_1 + X_2) \sim \text{Poisson}(\lambda_1 + \lambda_2)$. Applying this rule iteratively, we have that the arrival rates on Channel C_j follows $\text{Poisson}(\sum_i \lambda_{i,j} + \lambda_n)$.

The P2P streaming system is modeled as an $M/G/\infty$ system here. Thus, according to the queueing theory, the expected number of viewers is given by

$$\overline{N}_j = (\sum_i \lambda_{i,j} + \lambda_n)/\mu_j, \text{ when } i \neq j \tag{7.3}$$

Thus, the theorem follows. □

Suppose the lifetime of bootstrapping peers, denoted by 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 equal to F. Define the probability that a bootstrapping peer is in the network when it is called to serve other peers as *availability*.

Theorem 15. *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

$$\begin{aligned}\Pr\{L_s \geq t\} &= 1 - F_{L_s}(t;\gamma) \\ &= e^{-\gamma t},\end{aligned} \tag{7.4}$$

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

$$\Pr\{L_s \geq T_R + t | L_s > T_R\} = \Pr\{L_s \geq t\}. \tag{7.5}$$

Let T_R be update time when an agent is checking if the bootstrapping peer is still in the network. If some bootstrapping peers have left the network, an agent replaces them with new ones. According to Equation (7.5), once a bootstrapping peer is in the network on the update point, the probability that it stays for at least time t is as same as that of a newly joined bootstrapping peer. Thus, we can apply the same Equation (7.4) for all bootstrapping peers.

It is obvious that the time of a service call for a bootstrapping peer, i.e., t in Equation (7.4), is uniformly distributed in the time interval $[T_R, T_R + T_a]$ 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 8. *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 $(1 - e^{-\gamma_a/F_a})F_a/\gamma_a$.*

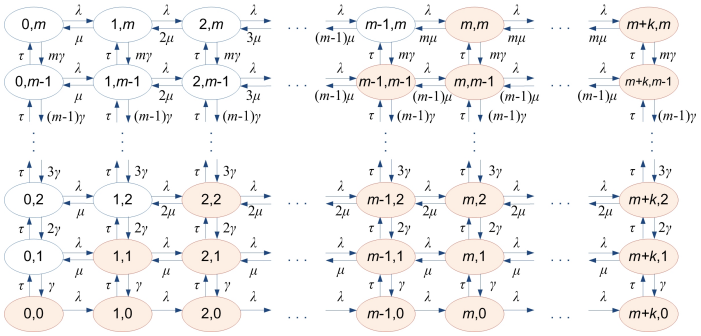


Figure 7.5: M/M/m/$m+k$ model

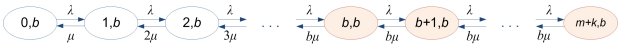


Figure 7.6: Isolated model on b^{th} level

Suppose that an 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 NAP, 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 7.5 illustrates our multi-level model, where the color-shaded states mean 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.

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. Because bootstrapping peers are distilled with superior lifetime expectation from general peers, it is reasonable to assume the arrival rate of general peers is much higher than the departure rate of bootstrapping peers, i.e. $\lambda \gg \gamma$. Thus, we can closely approximate the steady-state probabilities by the following strategy. First, we 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 7.6 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 7.7,

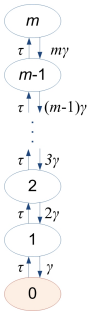


Figure 7.7: Single-column model

where each state (b) represents the number of available bootstrapping peers.

Theorem 16. *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 $(0,b),(1,b),...$ 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

$$\Pr(n,b) = \begin{cases} \frac{1}{n!}(\frac{\lambda}{\mu})^n \Pr(0,b) & \text{if } n \le b, \\ \frac{1}{b^{n-b}b!}(\frac{\lambda}{\mu})^n \Pr(0,b) & \text{if } n > b. \end{cases} \tag{7.6}$$

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 7.7 is a M/M/m/m queue. Applying the same method, we can obtain

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

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

$$\begin{aligned}\Pr(0,b) &= \frac{\Pr(b)}{\sum_{i=0}^{b} \frac{1}{i!}(\frac{\lambda}{\mu})^i + \sum_{i=b+1}^{m+k} \frac{1}{b^{i-b}b!}(\frac{\lambda}{\mu})^i}\\ &= \frac{(\frac{\tau}{\gamma})^b \Pr(0)}{b![\sum_{i=0}^{b} \frac{1}{i!}(\frac{\lambda}{\mu})^i + \sum_{i=b+1}^{m+k} \frac{1}{b^{i-b}b!}(\frac{\lambda}{\mu})^i]}\\ &= \frac{(\tau/\gamma)^b}{b![\sum_{i=0}^{m} \frac{1}{i!}(\frac{\tau}{\gamma})^i][\sum_{i=0}^{b} \frac{1}{i!}(\frac{\lambda}{\mu})^i + \sum_{i=b+1}^{m+k} \frac{1}{b^{i-b}b!}(\frac{\lambda}{\mu})^i]}.\end{aligned} \tag{7.7}$$

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 17. *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 (7.6) and (7.7), 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 18. *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 19. *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 17. *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 18. *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 a 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 20. *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 17 and 18, 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 are 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 21. *The expected churn-induced delay based on our scheme, denoted as $\overline{d}$, can be carried out by*

$$\overline{d} = \Pr(S)(\overline{d}_w+\overline{d}_f+\overline{d}_{pv})+(1-\Pr(S))\overline{d}_b, \tag{7.8}$$

where $\overline{d}_b = \overline{d}_p+\overline{d}_{sc}$ is the bootstrapping delay by contacting ordinary peers in the channel for connections, including the communication delay and the delay to get service from 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 an ordinary peer list from agents in the channel because this process runs preventively in the background of viewing current channel and makes no 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.

7.3 Performance Evaluation

In this section, we evaluate NAP against previous general scheme in terms of delay via theoretical experiments and simulation study.

7.3.1 Theoretical Evaluation

Based on the results from Section 7.2, we can numerically explore the performance of NAP by tuning some key parameters to the system.

To evaluate the performance of our scheme against the general scheme, we compare the delay occurring in NAP, expressed by Equation (7.8), with channel switching delay in the original scheme, i.e., Equation (7.1) and recovery delay in Equation (7.2). Besides $\overline{d}_A + \overline{d}_{ls}$, we may find the major difference between $\overline{d}_w + \overline{d}_f + \overline{d}_{pv}$ and $\overline{d}_b$, where NAP 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 the general scheme as *GS*. In the first experiment, we compare NAP with GS with varying rate of viewer arrivals to an agent. According to [5], we set the typical properties of a P2P streaming system as following: 1) the expected time a bootstrapping peer serves a viewer before handing it over 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 can manage is 100; 5) the number of viewers waiting 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 one per 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; $\overline{d}_{pv} = 0.5$ second; and $\overline{d}_f = 2$ seconds. For recovery delay, we change $\overline{d}_{sc}$ to 2

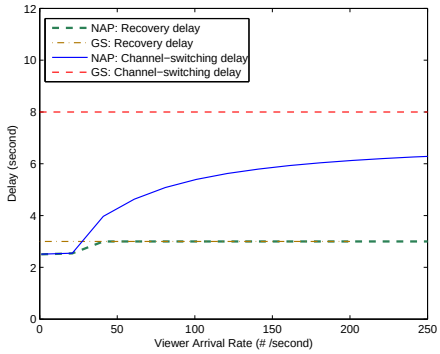


Figure 7.8: NAP v.s. General Scheme when viewer arrival rate changes.

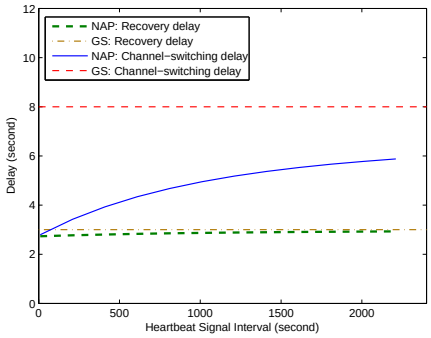
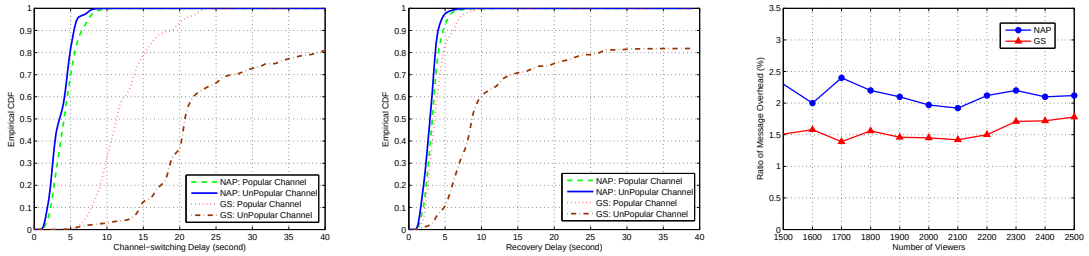


Figure 7.9: NAP v.s. General Scheme when heartbeat signal interval changes.



(a) CDF of Channel-switching Delay (b) CDF of Recovery Delay (c) Ratio of Message Overhead

Figure 7.10: Performance comparison between NAP and General Scheme (GS).

seconds since viewers in recovery has already contacted some peers for chunk information before interruption occurs.

From Figure 7.8, we can observe that NAP 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, NAP is better than GS when the arrival rates are lower than 40. When arrival rates go up, the performance of NAP almost converges to that of GS. This is because viewer waiting time is longer when bootstrapping peers are busy. We should notice the viewer also contacts ordinary peers simultaneously when they contact the agent for streaming service. This strategy make the worst-case performance of NAP at least equal to GS. To increase the performance, we can add more agents in the channel to keep the viewer arrival rate low.

Now, we compare NAP with GS from the perspective 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 7.9, we can also see NAP generally outperforms GS in terms of channel switching

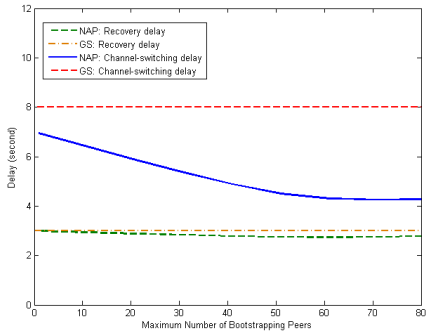


Figure 7.11: NAP v.s. General Scheme when the maximum number of bootstrapping peers changes.

delay. As of recovery delay, the difference is very close, about 12%. As we know, heartbeat signal checks 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 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, NAP can save 43% in channel-switching delay.

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 7.11, 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 NAP and GS in channel-switching delay is only 1.8 seconds at most.

From the above theoretical analysis, we observe that NAP significantly outperforms GS. To improve the performance of NAP, we can increase the number of agents or increase the heartbeat signal frequency.

7.3.2 Simulation Experiments

In this section, we compare the performance of NAP and GS via multiple simulations. Our simulation builds on P2P simulator [101], which was enhanced to support switching behaviors among multiple channels.

We simulate a live streaming session of 300 Kbps. From previous studies, we know that network bandwidth exhibits rich diversity [81]. Based on this, we set the upload capacity among peers as shown in Table 7.1. In our simulation, we set the peak channel popularity follow a Zipf distribution, which is presented in [5]. The streaming system is organized into

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

Table 7.1: Capacity Distribution on Peers

a mesh structure based on our previous work [29]. Peers come to the system in a Poisson process. After viewing a channel for period, peers may depart from the system, or switch to other channel with a probability proportional to the channel popularity. There are initially 50 streaming channels and 2,500 peers in the system.

In Figure 7.10a, we plot the empirical CDF of channel-switching delay under NAP and GS. *Popular channel* means switching to a channel with more than 100 viewers, while *unpopular channel* means switching to a channel with less than 10 viewers. From the figure, we observe NAP has much less channel-switching delay than GS in both popular and unpopular channels. Under GS, the delay in popular channel is significantly shorter than in unpopular channel. In contrast, NAP has a slightly better performance in unpopular channel than popular channel. It is because that under GS, bandwidth resource in unpopular channel can easily starve with less peers serving the channel, while under NAP, bootstrapping peers may be allocated to the unpopular channel with more than required bandwidth resources (higher than the actual popularity distribution). In Figure 7.10b, we illustrate similar experiments for recovery delay. It seems the results show a similar observation with the experiments for channel-switching delay except popular channel under GS has a closer CDF to the CDF curves under NAP.

Figure 7.10c shows the ratio of message overhead to the video traffic. We can see NAP has a slightly higher message overhead ratio than GS. This result indicates NAP can be widely used in the streaming system with feasible message overhead.

7.4 Summary

In this chapter, we develop NAP, a Novel Agent-based P2P scheme, to reduce churn-induced delays and a queueing model to analyze its performance. The experimental results indicate NAP can significantly reduce the churn-induced delays, especially the channel-switching delay. NAP proposes the idea of using agents to facilitate the bootstrapping process in channel switching and peer recovery. We focus on describing the methodology of NAP and establishing the analytical framework. In the practical deployment of NAP, we also need to consider the security issues that may incurred by introducing agents and bootstrapping peers. Future work can also focus on optimal admission control scheme if we assume VIP

viewers have the priority over normal viewers in agent service.

Chapter 8

Conclusions and Future Work

In this dissertation, we devise solutions to the problems of minimizing streaming delays and churn-induced delays. The problem spaces regarding reducing delays in P2P streaming, especially the theoretical part, remain a challenge and open in the P2P field. Motivated by this, our work is strongly based on theoretical analysis. The methods that we exploit in this dissertation extend to multiple directions regarding P2P streaming and optimization. Specifically, we investigate and build the theoretical foundation of this dissertation based on approximation algorithms, clustering, primal-dual schema, network flow optimization, queueing theory, and so on.

To achieve a tractable theoretical analysis, we start our study with assumptions on symmetric network models observing triangle property in Fastream-I. Then, in the design of Fastream-II, we relieve such assumptions for a more practical strategy. We also assume no network dynamics in the first stage of algorithm design. Although the assumption is strong in practical P2P applications, the value of this paper lies in the theoretical framework and analysis, which sheds light on the practical design. To reduce the complexity of the problem, we focus only on minimizing the communication delay. For packet scheduling, there exists a vast array of solutions. We propose a future study combining packet scheduling with our algorithms. Next, in the design of NAP, we consider reducing delays from network churns, which actually can be combined any of the Fastream algorithms to yield low-delay streaming.

In the following, we first summarize our work. Then, we propose future work in several directions based on the results of this dissertation.

8.1 Research Summary

Our work is motivated by our desire to minimize delays in current P2P streaming system. In this dissertation, we focus on four problems:

1) **Building maximum-delay-minimized overlay streaming mesh**. MDPS problem provides assurance of service quality. Theoretical works on the minimum delay P2P streaming problem are limited and remain open. Due to the lack of formal theoretical bounds, intuitions and heuristics have driven the design of P2P schemes so far [21, 54]. To address that, we formulated the minimum-delay P2P streaming problem, i.e. MDPS problem, and presented two solutions for it – a centralized approximation algorithm , and a distributed version. We show that our algorithms has a guaranteed performance bound. The distributed version has been extended to adapt 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) **Building average-delay-minimized overlay streaming mesh** [36, 45]. The MADPS problem is significantly different from MDPS problem. For example, the simulation results in [29] show that minimizing the maximum delay does not necessarily minimize the average end-to-end delay. Furthermore, Fastream-I assumes a network model with a symmetric graph, satisfying the triangle inequality. In contrast, we remove those assumptions in modeling the minimum average delay P2P streaming problem and present a near-optimal algorithm.

We consider the MADPS problem in the scenario that ISPs may provide viewers with diverse service options with different video quality, such as 720HD and 1080HD. Obtaining assurances on meeting the delay constraints in such heterogeneous network environments is a challenge. To solve this problem, we devise a streaming scheme which optimizes the bandwidth allocation to achieve the minimum average end-to-end P2P streaming delay. We first develop a generic analytical framework to model the minimum average delay P2P streaming problem. We then present Fastream-II to solve the MADPS problem. The core part of Fastream-II is devised as a fast approximation algorithm based on primal-dual schema. We know that any feasible solution to the dual problem provides an upper bound on the optimum. Thus, the approximation factor can be established by comparing the primal and dual solutions. In light of this, Fastream-II starts with a feasible solution for dual problem and relaxes the conditions for the primal problem. Then, Fastream-II iteratively improves the feasibility of the primal conditions and the optimality of the dual solution. Fastream-II winds up with feasible solutions for both primal and dual problems. So, the gap between them is the approximation factor.

3) **Building average-delay-minimized mesh in multi-channel streaming**. Minimizing streaming delay in multi-channel scenarios poses extra challenges . Traditionally, multi-channel streaming systems organize peers into multiple overlays, where each overlay is formed

by a swarm of peers viewing the same channel [8,22,37]. Peers in the same overlay exchange data exclusively inside the swarm. We call such a scheme of isolated multiple overlays IMO. However, the frequent channel switching (or channel churn) undermines the viewing reliability under the IMO model. Furthermore, the upload bandwidth resources of different channels are highly unbalanced, where channels with low bandwidth suffer poor performance. (The IMO model does not enable bandwidth sharing from channels with surplus bandwidth.) Thus, minimizing multi-channel delays based on the IMO model cannot provide an effective solution. In other words, once any viewer changes the channel, the minimum-delay streaming topology needs to change accordingly.

We find that VUD-based systems provide robustness in the high-channel-churn environments and flexible cross-channel resource sharing capability. In light of these promising advantages, we study the minimum-delay problem under the VUD-based model. The problem of how to share the bandwidth across different channels toward minimizing the average source-to-port streaming delay is an open problem. To solve the MSPDS problem, we devise an approximation algorithm: Fastream-III.

4) **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 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. However, maintaining preventive connections for all peers to all channels makes it impractical in terms of the enormous number of control signals and high message overhead. To obtain 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, agents 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) Fastream-I [29]: to the best of our knowledge, Fastream-I represents the first approximation algorithm that optimizes P2P streaming delay and provides a provable upper bound of $O(\sqrt{\log n})$. 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 $O(\sqrt{\log n})$. 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.

(2) Fastream-II [36, 45]: for a feasible solution, we start with the assumption of a static network—i.e., no churn. In this way, we can devise a framework which is analytically achiev-

able. The method will be most suitable for the scenario where a service provider deploys a set-top box at viewers' homes. In that case, even when a viewer turns off the TV, the set-top box can still contribute its bandwidth to other viewers. For this scenario, we first develop an analytical model that formulates the MADPS problem as an optimization problem. Then we propose Fastream-II to solve MADPS problem. Inspired by the primal-dual schema, we develop an approximation algorithm as the core of Fastream-II, called APX-Fastream-II, for optimally utilizing the bandwidth among peers subscribing to different video qualities, while achieving the minimum average streaming delay. We show that APX-Fastream-II's performance in terms of delay and running time is bounded. We show that there exists a trade-off between iStream-APX's approximation factor ω and its running time. The approximation factor is adjustable in the range of $(1, n]$, where n is the number of peers in the network. This trade-off allows users to flexibly tune the performance bound according to running time requirements. To the best of our knowledge, Fastream-II represents the first scheme towards MADPS problem based on approximation algorithm.

(3) Fastream-III: We present a polynomial-time algorithm – Fastream-III – for the MSPDS problem. We develop a tractable analytical framework for the MSPDS problem, which can provide an analytical foundation for algorithm design. We devise the critical part of Fastream-III by a fast approximation algorithm based on primal-dual schema, called APX-Fastream-III. One of the steps in APX-Fastream-III involves shortest path search. We extend the shortest path search from each step for just searching one node in Fastream-II to each step for building a shortest path tree to all nodes. This improvement greatly facilitates the algorithm efficiency. Finally, we show that the performance of APX-Fastream-III is also bounded by a factor of $1 + \omega$, where ω is an input parameter. We show that there exists a trade-off between the APX-Fastream-III's approximation factor ω and its running time, which we can tune the appropriate performance bound according the requirement on running time. To the best of our knowledge, we are not aware of any other solution to the MSPDS problem.

(4) NAP [40]: NAP is the first scheme for reducing the churn-induced delay with analytical model. Our scheme uses 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 an agent, which facilitates the bootstrapping process in channel switching and peer recovery. We build a queueing theory model for NAP. Based on this model, we theoretically analyze the performance of our scheme. We analyze the scheme's performance and derive the scheme's optimal settings based on the numerical experiments. Our numerical experiments indicate our scheme can significantly reduce the churn-induced delay, especially for the channel-switching delay.

8.2 Remarks and Discussion

Fastream algorithms provide the first approximation solutions to the minimum streaming delay problems. In contrast to past work in the field, our solutions provide performance bound in terms of delay and guarantee a bound on the polynomial running time. Our work paves a theoretical foundation for practical implementations of P2P live streaming systems , and reveals important factors in the design of a live streaming system. There are still further improvements required before these theory-oriented solutions can be employed in practical implementations. In the design of Fastream-I, we assume a symmetric network following the triangle inequality. In the current Internet , such assumptions may not hold. For example, streaming delay from node i to node j may differ from the delay from node j to node i due to different routing paths in the two directions and different queueing delays from underlying physical networks. Similarly, the triangle inequality may not hold in the practical system. In order to relieve such limitations, cross layer design provides a possible direction, where influences from the underlying physical networks will be considered with the overlay design. Also, real-time packet scheduling on nodes should be designed to dispatch packets according the delay requirement on the system. For tractable solutions, Fastream algorithms also assume static networks. Such design will be most suitable for systems built on set-top boxes that will continue serving the network even if the viewer turns off the TV. Considering general networks, Fastream algorithms need improvements on how to handle the network dynamics and keep the performance bounds at the same time. To deal with that, random graph theory may be exploited in the system design, where probability theory will be introduced to model the network dynamics.

Moreover, we build NAP on a dynamic system without constraints on network symmetry and triangle inequality. Therefore, NAP provides a practical solution towards reducing churn-induced delays. At the same time, we build a queueing model for NAP. With a parameterized study on this model, researchers can easily understand the key factors in optimizing such system. In a practical implementation, we should consider the security issues. For example, when we select agents, how can we avoid virus spread and peeks on personal information on those agents? Thus, security studies can be pursued to solve such practical concerns. Moreover, agents play a key role in NAP. What incentives should we provide for those agents so they can stay in network longer and provide more resources? To solve this, game theory may be investigated to enhance the performance.

8.3 Future Work

Based on our current direction of research, we propose the following highest priority future work:

Minimize the end-to-end streaming delay in multi-channel scenario. In the MSPDS problem,

we consider minimizing the average delay from source nodes to ports. Here, we propose to devise an algorithm that works towards minimizing end-to-end delay in multi-channel scenarios . Although end-to-end delay is highly dynamic due to channel-switching churns, we can still consider the minimum end-to-end delay problem with probabilistic estimation. We can assume the viewers distribution on different channels and the time they keep watching the channel. By probabilistic estimation, we can devise a strategy to select ports such that the mean end-to-end delay (source-to-port delay plus estimated port-to-viewer delay) can be minimized. By leveraging our current approximation algorithms with probability theory, we may guarantee an approximation bound on the mean end-to-end delay.

Design a delay-optimized admission control algorithms. Industrial deployment of P2P streaming application may involve different types of peers, including free user, basic service member paying fees to watch specific channels, and VIP members paying extra fees for high-quality service. However,the traditional scheme in cable TV to prioritize VIP members when bandwidth resource is not sufficient is not applicable. Some free users may have usable bandwidth to contribute. If we simply put them at the edge of the streaming mesh with worst service, we may lose the opportunity to rationally utilize this spare 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 interested in solving the following three problems:

Implement our algorithms and test it in PlanetLab. Simulation cannot completely reflect the influential factors in the real-world network systems. PlanetLab has emerged as a real-world test platform exclusively for the development of new network services [102]. PlanetLab currently consists of 1098 nodes at 511 sites. It provides a platform to develop new technologies for distributed storage, network mapping, peer-to-peer systems, distributed hash tables, and query processing. Implementations based on such real-world platform can help evaluate the performance of our distributed schemes with practical environments, and may help us identify some improvement directions on the schemes.

Minimize the scheduling delay algorithms. P2P streaming needs to exchange data chunks and decide which chunk should be transfered with priority. Delayed receiving of chunks 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 existing chunks 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 for exchanging agent data and to search for resources. We propose to design an improved gossip scheme to ensure satisfaction of 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. lower probability of departing , 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? [peer-to-peer multimedia streaming]," *Communications Magazine, IEEE*, vol. 45, no. 6, pp. 86 –92, Jun. 2007.

[2] R. Kumar, Y. Liu, and K. Ross, "Stochastic fluid theory for P2P streaming systems," in *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, May. 2007, pp. 919 –927.

[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*, Apr. 2008, pp. 1373 –1381.

[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*, Sep. 2008, pp. 289 –298.

[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, pp. 1672 –1687, Dec. 2007.

[6] X. Liao, H. Jin, Y. Liu, L. M. Ni, and D. Deng, "Anysee: Peer-to-peer live streaming," in *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, Apr. 2006, pp. 1 –10.

[7] G. Dán and V. Fodor, "Delay bounds and scalability for overlay multicast," in *NETWORKING 2008 Ad Hoc and Sensor Networks, Wireless Networks, Next Generation Internet*. Springer, Berlin Heidelberg, 2008, pp. 227–239.

[8] D. Wu, C. Liang, Y. Liu, and K. Ross, "View-upload decoupling: A redesign of multi-channel p2p video systems," in *INFOCOM 2009, IEEE*, Apr. 2009, pp. 2726 –2730.

[9] 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, Feb. 2008.

[10] PPLive. [Online]. Available: www.pplive.com

[11] PPS. [Online]. Available: www.pps.com

[12] Sopcast. [Online]. Available: www.sopcast.com

[13] D. Wu, Y. Liu, and K. Ross, "Queuing network models for multi-channel p2p live streaming systems," in *INFOCOM 2009, IEEE*, Apr. 2009, pp. 73 –81.

[14] 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*. New York, NY, USA: ACM, 2008, pp. 313–324.

[15] N. Parvez, C. Williamson, A. Mahanti, and N. Carlsson, "Analysis of bittorrent-like protocols for on-demand stored media streaming," in *SIGMETRICS '08: Proceedings of the 2008 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*. New York, NY, USA: ACM, 2008, pp. 301–312.

[16] F. Picconi and L. Massoulie, "Isp friend or foe? making p2p live streaming isp-aware," in *Distributed Computing Systems, 2009. ICDCS '09. 29th IEEE International Conference on*, Jun. 2009, pp. 413 –422.

[17] L. Walker, "Uncle sam wants napster!" p. E01, Nov. 8 2001. [Online]. Available: http://www.washingtonpost.com/ac2/wp-dyn?pagename=article&node=washtech/techthursday/columns/dotcom&contentId=A59099-2001Nov7

[18] V. Venkataraman, P. Francisy, and J. Calandrino, "Chunkyspread: Multitree unstructured peer-to-peer multicast," in *IPTPS '06*, 2006.

[19] N. Magharei, R. Rejaie, and Y. Guo, "Mesh or multiple-tree: A comparative study of live P2P streaming approaches," in *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, May. 2007, pp. 1424 –1432.

[20] 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*, Apr. 2008, pp. 473 –476.

[21] D. Ren, Y.-T. Li, and S.-H. Chan, "On reducing mesh delay for peer-to-peer live streaming," in *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, Apr. 2008, pp. 1058 –1066.

[22] 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, pp. 356 –361.

[23] N. Magharei and R. Rejaie, "Prime: Peer-to-peer receiver-driven mesh-based streaming," *Networking, IEEE/ACM Transactions on*, vol. 17, no. 4, pp. 1052 –1065, Aug. 2009.

[24] N. Magharei, Y. Guo, and R. Rejaie, "Issues in offering live P2P streaming service to residential users," in *Consumer Communications and Networking Conference, 2007. CCNC 2007. 4th IEEE*, Jan. 2007, pp. 757 –762.

[25] J. Wang, C. Qiao, Y. Li, and K. Lu, "Minimizing the worst-case playback delay in vod services over passive optical networks," in *INFOCOM, 2010 Proceedings IEEE*, Mar. 2010, pp. 1 –9.

[26] T. Small, B. Liang, and B. Li, "Scaling laws and tradeoffs in peer-to-peer live multimedia streaming," in *MULTIMEDIA '06: Proceedings of the 14th annual ACM international conference on Multimedia*. New York, NY, USA: ACM, 2006, pp. 539–548.

[27] T. Silverston, O. Fourmaux, and J. Crowcroft, "Towards an incentive mechanism for peer-to-peer multimedia live streaming systems," in *Peer-to-Peer Computing , 2008. P2P '08. Eighth International Conference on*, Sep. 2008, pp. 125 –128.

[28] F. Hecht, T. Bocek, C. Morariu, D. Hausheer, and B. Stiller, "Liveshift: Peer-to-peer live streaming with distributed time-shifting," in *Peer-to-Peer Computing , 2008. P2P '08. Eighth International Conference on*, Sep. 2008, pp. 187 –188.

[29] F. Huang, B. Ravindran, and V. Kumar, "An approximation algorithm for minimum-delay peer-to-peer streaming," in *Peer-to-Peer Computing, 2009. P2P '09. IEEE Ninth International Conference on*, Sep. 2009, pp. 71 –80.

[30] Z. Liu, Y. Shen, S. S. Panwar, K. W. Ross, and Y. Wang, "Using layered video to provide incentives in p2p live streaming," in *P2P-TV '07: Proceedings of the 2007 workshop on Peer-to-peer streaming and IP-TV*. New York, NY, USA: ACM, 2007, pp. 311–316.

[31] A. T. Nguyen, B. Li, and F. Eliassen, "Chameleon: Adaptive peer-to-peer streaming with network coding," in *INFOCOM, 2010 Proceedings IEEE*, Mar. 2010, pp. 1 –9.

[32] V. Gopalakrishnan, R. Jana, R. Knag, K. Ramakrishnan, D. Swayne, and V. Vaishampayan, "Characterizing interactive behavior in a large-scale operational iptv environment," in *INFOCOM, 2010 Proceedings IEEE*, Mar. 2010, pp. 1 –5.

[33] Z. Li, D. Tsang, and W.-C. Lee, "Understanding sub-stream scheduling in p2p hybrid live streaming systems," in *INFOCOM, 2010 Proceedings IEEE*, Mar. 2010, pp. 1 –5.

[34] D.-C. Tomozei and L. Massoulie, "Flow control for cost-efficient peer-to-peer streaming," in *INFOCOM, 2010 Proceedings IEEE*, Mar. 2010, pp. 1 –9.

[35] 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.

[36] F. Huang, M. Khan, and B. Ravidran, "On minimizing average end-to-end delay in p2p live streaming systems," in *OPODIS'10: 14th International Conference On Principles Of Distributed Systems*, Dec. 2010.

[37] Y. Ma, R. Chbeir, K. Yetongnon, and G. Su, "A multi-channel end system multicasting scheme on p2p cluster overlay," in *Signal-Image Technologies and Internet-Based System, 2007. SITIS '07. Third International IEEE Conference on*, Dec. 2007, pp. 305 –312.

[38] 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*, Apr. 2008, pp. 1031 –1039.

[39] D. Choffnes, M. Sanchez, and F. Bustamante, "Network positioning from the edge - an empirical study of the effectiveness of network positioning in p2p systems," in *INFOCOM, 2010 Proceedings IEEE*, Mar. 2010, pp. 1 –5.

[40] F. Huang, B. Ravindran, and M. Khan, "Nap: An agent-based scheme on reducing churn-induced delays for p2p live streaming," in *Peer-to-Peer Computing '10*, 2010.

[41] 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.

[42] B. Mitra, A. Dubey, S. Ghose, and N. Ganguly, "How do superpeer networks emerge?" in *INFOCOM, 2010 Proceedings IEEE*, Mar. 2010, pp. 1 –9.

[43] E. Kurutepe and T. Sikora, "Multi-view video streaming over P2P networks with low start-up delay," in *Image Processing, 2008. ICIP 2008. 15th IEEE International Conference on*, Oct. 2008, pp. 3088 –3091.

[44] F. Huang, K. Han, B. Ravindran, and E. Jensen, "Integrated real-time scheduling and communication with probabilistic timing assurances in unreliable distributed systems," in *Engineering of Complex Computer Systems, 2008. ICECCS 2008. 13th IEEE International Conference on*, Mar. 2008, pp. 79 –88.

[45] F. Huang, M. Khan, and B. Ravidran, "Technical report: On minimizing average end-to-end delay in p2p live streaming systems," Virginia Tech, Tech. Rep., 2010. [Online]. Available: http://staff.vbi.vt.edu/maleq/papers/APX_P2P_average_delay.pdf

[46] Q. Wang, L. Vu, K. Nahrstedt, and H. Khurana, "Mis: Malicious nodes identification scheme in network-coding-based peer-to-peer streaming," in *INFOCOM, 2010 Proceedings IEEE*, Mar. 2010, pp. 1 –5.

[47] H. Yin, C. Lin, Q. Zhang, Z. Chen, and D. Wu, "Truststream: A secure and scalable architecture for large-scale internet media streaming," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 18, no. 12, pp. 1692 –1702, Dec. 2008.

[48] S. Asaduzzaman, Y. Qiao, and G. Bochmann, "Cliquestream: An efficient and fault-resilient live streaming network on a clustered peer-to-peer overlay," in *Peer-to-Peer Computing , 2008. P2P '08. Eighth International Conference on*, Sep. 2008, pp. 269 –278.

[49] 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*, Sep. 2008, pp. 279 –288.

[50] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O'Toole, Jr., "Overcast: reliable multicasting with on overlay network," in *OSDI'00: Proceedings of the 4th conference on Symposium on Operating System Design & Implementation*. Berkeley, CA, USA: USENIX Association, 2000, pp. 14–14.

[51] Y. Liu, "Delay bounds of chunk-based peer-to-peer video streaming," *Networking, IEEE/ACM Transactions on*, vol. 18, no. 4, pp. 1195 –1206, Aug. 2010.

[52] 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, pp. 2556 –2560.

[53] D. Ren, Y.-T. Li, and S.-H. Chan, "Fast-mesh: A low-delay high-bandwidth mesh for peer-to-peer live streaming," *Multimedia, IEEE Transactions on*, vol. 11, no. 8, pp. 1446 –1456, Dec. 2009.

[54] G. Bianchi, N. Blefari Melazzi, L. Bracciale, F. Lo Piccolo, and S. Salsano, "Streamline: An optimal distribution algorithm for peer-to-peer real-time streaming," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 21, no. 6, pp. 857 –871, Jun. 2010.

[55] 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*. New York, NY, USA: ACM, 2007, pp. 127–136.

[56] 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.

[57] 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*, vol. 18, no. 12, pp. 1663 –1674, Dec. 2007.

[58] C. Wu, B. Li, and S. Zhao, "Multi-channel live p2p streaming: Refocusing on servers," in *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, Apr. 2008, pp. 1355 –1363.

[59] 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*. New York, NY, USA: ACM, 2008, pp. 325–336.

[60] G. Ezovski, A. Tang, and L. Andrew, "Minimizing average finish time in p2p networks," in *INFOCOM 2009, IEEE*, Apr. 2009, pp. 594 –602.

[61] R. Kumar, Y. Liu, and K. Ross, "Stochastic fluid theory for P2P streaming systems," in *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, May. 2007, pp. 919 –927.

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

[63] 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.

[64] 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.

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

[66] 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.

[67] N. Garg and J. Könemann, "Faster and simpler algorithms for multicommodity flow and other fractional packing problems," *SIAM J. Comput.*, vol. 37, no. 2, pp. 630–652, 2007.

[68] G. Karakostas, "Faster approximation schemes for fractional multicommodity flow problems," *ACM Trans. Algorithms*, vol. 4, no. 1, pp. 1–17, 2008.

[69] 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, pp. 190 –200.

[70] 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's surfing behavior and preference," in *Broadband Multimedia Systems and Broadcasting, 2008 IEEE International Symposium on*, Mar. 2008, pp. 1 –6.

[71] 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*, vol. 2, 2004, pp. 971 – 975.

[72] 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, Jun. 2008.

[73] H. Fuchs and N. Farber, "Optimizing channel change time in iptv applications," in *Broadband Multimedia Systems and Broadcasting, 2008 IEEE International Symposium on*, Mar. 2008, pp. 1 –8.

[74] K. Xu, M. Zhang, J. Liu, Z. Qin, and M. Ye, "Proxy caching for peer-to-peer live streaming," *Computer Networks*, 2009.

[75] 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.

[76] B. Cheng, X. Liu, Z. Zhang, and H. Jin, "A measurement study of a peer-to-peer video-on-demand system," in *IPTPS '07*, 2007.

[77] 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, pp. 1 –8.

[78] A. Binzenhofer, D. Staehle, and R. Henjes, "On the stability of chord-based P2P systems," in *Global Telecommunications Conference, 2005. GLOBECOM '05. IEEE*, vol. 2, Nov. 2005, p. 5 pp.

[79] 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*, Apr. 2006, pp. 1 –11.

[80] 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, Jun. 2007.

[81] 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.

[82] V. V. Vazirani, *Approximation Algorithm*. New York: Springer-Verlag New York, LLC, 2007.

[83] M. Wang, L. Xu, and B. Ramamurthy, "Linear programming models for multi-channel p2p streaming systems," in *INFOCOM, 2010 Proceedings IEEE*, Mar. 2010, pp. 1 –5.

[84] Z. Liu, C. Wu, B. Li, and S. Zhao, "Distilling superior peers in large-scale p2p streaming systems," in *INFOCOM 2009, IEEE*, Apr. 2009, pp. 82 –90.

[85] 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.

[86] 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.

[87] 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.

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

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

[90] B. Awerbuch, "Randomized distributed shortest paths algorithms," in *STOC '89: Proceedings of the twenty-first annual ACM symposium on Theory of computing*. New York, NY, USA: ACM, 1989, pp. 490–500.

[91] W. T. Zaumen and J. J. Garcia-Luna Aceves, "Dynamics of distributed shortest-path routing algorithms," *SIGCOMM Comput. Commun. Rev.*, vol. 21, no. 4, pp. 31–42, 1991.

[92] P. Humblet, "Another adaptive distributed shortest path algorithm," *Communications, IEEE Transactions on*, vol. 39, no. 6, pp. 995 –1003, Jun. 1991.

[93] N. Garg and J. Koenemann, "Faster and simpler algorithms for multicommodity flow and other fractional packing problems." in *FOCS '98: Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, 1998.

[94] Y. Liu, L. Guo, F. Li, and S. Chen, "A case study of traffic locality in internet p2p live streaming systems," in *Distributed Computing Systems, 2009. ICDCS '09. 29th IEEE International Conference on*, Jun. 2009, pp. 423 –432.

[95] H. Elaarag, "Improving tcp performance over mobile networks," *ACM Comput. Surv.*, vol. 34, no. 3, pp. 357–374, 2002.

[96] 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*, Apr. 2008, pp. 342 –350.

[97] C. Wu, B. Li, and S. Zhao, "Multi-channel live p2p streaming: Refocusing on servers," in *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, Apr. 2008, pp. 1355 –1363.

[98] F. Huang, B. Ravindran, and E. Jensen, "Rt-p2p: A scalable real-time peer-to-peer system with probabilistic timing assurances," in *Embedded and Ubiquitous Computing, 2008. EUC '08. IEEE/IFIP International Conference on*, vol. 1, Dec. 2008, pp. 97 –103.

[99] A. Ganesh, A.-M. Kermarrec, and L. Massoulie, "Peer-to-peer membership management for gossip-based protocols," *Computers, IEEE Transactions on*, vol. 52, no. 2, pp. 139 – 149, Feb. 2003.

[100] I. Zubair and M. Islam, "Adaptive trust management in P2P networks using gossip protocol," in *Emerging Technologies, 2008. ICET 2008. 4th International Conference on*, Oct. 2008, pp. 176 –181.

[101] M. Zhang, "Peer-to-peer streaming simulator," http://media.cs.tsinghua.edu.cn/~zhangm/download.

[102] PlanetLab. [Online]. Available: www.planet-lab.org