EFFICIENT ALGORITHMS FOR MULTI-SENDER DATA TRANSMISSION IN SWARM-BASED PEER-TO-PEER STREAMING SYSTEMS

by

Yuanbin Shen
B.Eng., University of Science and Technology of China, Hefei, China, 2008

A Thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science
in the School
of
Computing Science

SIMON FRASER UNIVERSITY
Summer 2010

APPROVAL

Name:	Yuanbin Shen
Degree:	Master of Science
Title of Thesis:	Efficient Algorithms for Multi-sender Data Transmission in Swarm-based Peer-to-peer Streaming Systems
Examining Committee:	Dr. Kay C. Wiese Chair
	Dr. Mohamed Hefeeda, Senior Supervisor
	Dr. Joseph G. Peters, Supervisor
	Dr. Qianping Gu, SFU Examiner
Date Approved:	August 17, 2010

Abstract

In peer-to-peer (P2P) swarm-based streaming systems, each video sequence is typically divided into segments, which are then streamed from multiple senders to a receiver. The receiver needs to coordinate the senders by specifying a transmission schedule for each of them. Our goal is to compute the transmission schedules in order to maximize the video quality at the receivers. We prove that this problem is NP-Complete. We present an Integer Linear Programming (ILP) formulation for this problem. This optimal solution, however is computationally expensive and is not suitable for real-time streaming systems. Thus, we propose an approximation algorithm to solve this problem, which has an approximation factor of 2. We implement the proposed algorithm in a P2P simulator and in a P2P prototype system. Our evaluation results indicate that the proposed algorithm outperforms other algorithms deployed in current systems.

Keywords: peer-to-peer; video streaming; segment transmission scheduling;

Acknowledgments

First of all, I would like to express my deepest gratitude to my senior supervisor, Dr. Mohamed Hefeeda, for mentoring me in the past two years. His encouragement and enthusiasm in research have always propelled me to strive for excellence. The ways of thinking and doing research that I have learned from our numerous times of discussions have become precious treasures in my life. This thesis would not have been possible without him.

I then would like to show my gratitude to my supervisor Dr. Joseph G. Peters, and my thesis examiner Dr. Qianping Gu for serving on my committee and reviewing my thesis, and Dr. Kay C. Wiese, for taking time to chair my thesis defense. I would also like to thank all the faculty and staff in the School of Computing Science, Simon Fraser Univeristy, especially: Dr. Ramesh Krishnamurti, Dr. Funda Ergun, Dr. Tamara Smyth and Dr. Oliver Schulte for what I have learned from their courses.

I am grateful to all the members at the Network Systems Lab for their friendship and fun made in my life, especially: Cheng-Hsin Hsu, Yi Liu, Kianoosh Mokhtarian, Ahmed Hamza, Somsubhra Sharangi, Cong Ly, Shabnam Mirshokraie and Jeffrey Gao. A big thank-you goes to Cheng-Hsin Hsu, who has helped me a lot for my academic work, and from whom I learned hard working and efficiency.

Finally, I am indebted to my family for their love and support. This thesis is dedicated to them.

Contents

Approval ii

Abstract iii

Acknowledgments iv

Contents v

List of Tables vii

List of Figures viii

1 Introduction 1

1.1 Introduction 1

1.2 Problem Statement and Thesis Contributions 2

1.3 Thesis Organization 3

2 Background and Related Work 4

2.1 Overview of P2P Systems 4

2.2 Overlay Networks 5

2.3 P2P Applications 6

2.3.1 P2P Computing 6

2.3.2 P2P Storage 6

2.3.3 P2P Media Streaming 7

2.4 P2P Media Streaming Systems 7

2.4.1 Live and On-demand Streaming 7

2.4.2 Tree-based and Swarm-bashed Structures 8

2.4.3 Push-based and Pull-based Protocols 8
2.4.4 Peer Matching 9
2.4.5 Segment Transmission Scheduling 9
2.5 Related Work 10

3 Segment Transmission Scheduling Problem 12
3.1 System Model 12
3.2 Problem Statement 15
3.3 Hardness 16

4 Proposed Approximation Algorithm 17
4.1 Problem Formulation and Optimal Solution 17
4.2 Overview of the Proposed Algorithm 21
4.3 Approximation Factor and Time Complexity 23
4.4 Discussion 24

5 Evaluation Using Simulation 26
5.1 Simulation Setup 26
5.2 Simulation Results 29

6 Evaluation Using Prototype Implementation 35
6.1 Experimental Setup 35
6.2 Experimental Results 39

7 Conclusions and Future Work 40
7.1 Conclusions 40
7.2 Future Work 41

Bibliography 42

List of Tables

3.1 List of symbols used in the thesis. 13

5.1 List of video parameters used in the thesis. 27
5.2 Peer uploading bandwidth distribution. 27

List of Figures

3.1 Two sliding windows in the system. 13

4.1 The proposed approximation algorithm SSTF. 22

5.1 Overall comparison of the proposed (SSTF) algorithm and the Rarest First (RF) algorithm used in several deployed systems and the Min-Cost flow based (MC) algorithm and the Weighted Segment Scheduling (WSS) algorithm recently proposed in the literature and the optimal algorithm (OPT) solved directly from the ILP formulation. (Terminator 2 video) 29

5.2 Load balance and running time of the WSS, SSTF, RF, MC and OPT algorithms. Note: the running time curves for the SSTF and RF algorithms are not visible in the figure because they are very close to 0. (Terminator 2 video) 30

5.3 Overall comparison of the proposed (SSTF) algorithm and the Rarest First (RF) algorithm used in several deployed systems and the Min-Cost flow based (MC) algorithm and the Weighted Segment Scheduling (WSS) algorithm recently proposed in the literature and the optimal algorithm (OPT) solved directly from the ILP formulation. (SonyDemo video) 30

5.4 Load balance and running time of the WSS, SSTF, RF, MC and OPT algorithms. Note: the running time curves for the SSTF and RF algorithms are not visible in the figure because they are very close to 0. (SonyDemo video) . 31

5.5 Segment level comparison of the proposed (SSTF) algorithm and the Rarest First (RF) algorithm used in several deployed systems and the Min-Cost flow based (MC) algorithm and the Weighted Segment Scheduling (WSS) algorithm proposed in the literature and the optimal algorithm (OPT) solved directly from the ILP formulation. (Terminator 2 video) 34

6.1 A high level diagram for the prototype P2P streaming system implementation. 36
6.2 Overall comparison in PlanetLab-based experiments. (SonyDemo video) . . . 38
6.3 Load balance and running time in PlanetLab-based experiments. (SonyDemo video) . . . 38

Chapter 1

Introduction

In this chapter, we introduce the segment transmission scheduling problem in Peer-to-Peer (P2P) streaming systems that we solve in this thesis and we summarize the contributions of our work.

1.1 Introduction

P2P streaming systems [1–4] have been proposed to distribute multimedia contents at low infrastructure costs. They can be built in two different ways [5]: (i) tree-based systems in which one or more trees are constructed to connect peers for transferring contents [6–8], and (ii) swarm-based systems in which each peer connects to a few neighboring peers without an explicit network topology before exchanging data with each other [1, 9, 10]. We consider swarm-based systems, because they incur lower maintenance overheads, adapt better to network dynamics, are easier to implement [11], and lead to better perceived video quality [12].

In swarm-based systems, a video sequence is partitioned into small *segments*, and segments are transmitted from multiple senders to a receiver. The receiving peer must coordinate the segment transmission from its senders. More precisely, a receiver runs a *scheduling* algorithm to compose a transmission schedule for its senders, which specifies for each sender the assigned segments and their transmission times.

Previous works [13, 14] show that when resources (especially bandwidth) are enough to stream videos in a P2P streaming system, almost all existing scheduling algorithms perform equally well. While with the advent of more and more high quality video streams over the Internet, resources are never sufficient, thus more intelligent scheduling algorithms are needed to fully utilize those limited resources. Composing segment transmission schedules is not an easy task, as P2P streaming systems impose time constraints on segment transmission. Segments arriving at the receiver after their decoding deadlines are not useful, because they cannot be rendered to users for improving video quality. Hence, segment scheduling algorithms should strive to maximize the perceived video quality delivered by the on-time segments, which refer to those segments that meet their decoding deadlines. Optimally constructing segment schedules is computationally expensive [9], and thus existing systems either resort to simple heuristic algorithms [9,10,15], or assign each segment an ad-hoc *utility function* and solve the simplified problem of maximizing the system-wide utility [16,17]. These algorithms provide *no* performance guarantees on the number of on-time delivered segments, and may result in playout glitches and degraded video quality. A recent work [11] pointed out that these existing algorithms might work in live streaming systems as peers in these systems share a small scheduling window and are less sensitive to the performance of scheduling algorithms; however, they may not work well in on-demand streaming systems, especially when the system bandwidth resources are limited to satisfy all requests from all peers.

1.2 Problem Statement and Thesis Contributions

We study the problem of transmitting a video stream from multiple senders to a receiver in a P2P streaming system. This stream is divided into segments, which can contain one or more video frames. We consider a general P2P streaming system that aggregates any number of coded frames in each segment. Segments have different sizes because coded frames in video streams typically vary in sizes. The receiver monitors its senders in terms of segment availability and uploading bandwidth. Given this information, the problem is to compute a segment transmission schedule for each scheduling window at the receivers, which specifies from which senders to request which part of the data. Our goal is to construct such schedules to maximize the number of segments that arrive on time at the receivers in order to improve the video quality rendered to the users. We give a formal description of the problem in Chapter 3.

We solve the problem of segment transmission scheduling in both live and on-demand P2P streaming systems. In particular, the contributions of this thesis can be summarized as follows [18]:

- We propose an Integer Linear Programming (ILP) formulation for the segment transmission scheduling problem so that it can be optimally solved using any existing ILP solvers. However, directly solving the ILP problem may take prohibitively long time and is not suitable for real-time P2P streaming systems.
- We propose an efficient approximation algorithm, which constructs transmission schedules with performance *guarantees* on the number of on-time segments. We show that the algorithm has an approximation factor of 2.
- We analyze the time complexity of the proposed approximation algorithm, and show that it runs very fast, in polynomial time.
- We implement the proposed algorithm in a simulator for dynamic P2P systems. We also implement recent scheduling algorithms used in current P2P systems and proposed in the literature, and we compare our algorithm against them. The simulation results show that our proposed algorithm consistently outperforms other scheduling algorithms used in current systems and proposed in the literature.
- We design and implement a prototype P2P streaming system in which we implement the proposed algorithm and other scheduling algorithms. We deploy the system on 500 nodes of the PlanetLab [19] wide area testbed, and we conduct extensive experiments with multiple videos that have diverse characteristics. Our experimental results further confirm that our algorithm outperforms the other algorithms.

1.3 Thesis Organization

The rest of this thesis is organized as follows. Chapter 2 provides some background on P2P systems and their applications, especially P2P media streaming applications. It also summarizes the related work of the considered scheduling problem in the literature. In Chapter 3, we describe the considered system model and the hardness of the segment transmission schedules problem. We then formulate and solve the problem in Chapter 4. Evaluation using simulation of the proposed algorithm is given in Chapter 5, and using actual implementation in Chapter 6. We conclude this thesis in Chapter 7.

Chapter 2

Background and Related Work

In this chapter, we first give a brief introduction to P2P systems and their applications, especially P2P media streaming applications. We then investigate the current state of research on the segment transmission scheduling problem in P2P streaming systems proposed in the literature.

2.1 Overview of P2P Systems

P2P systems are distributed systems in which peers interact with each other without the need of centralized coordination systems. They are popularized mainly by P2P file sharing systems, such as Bittorrent [20], but now have been used in a wide range of applications, including distributed computing [21,22], networked storage systems [23,24], and media streaming [25, 26]. The design and organization of P2P networks also inspired structures outside the computer science area, including social networks like Facebook [27] and Consumer-to-Consumer e-commerce like eBay [28]. The power of a P2P system comes from its utilization of resources on large amount of end users, as opposed to traditional client-server network structures. In a client-server network model, all clients make requests to use resources (such as bandwidth and CPU time) on a server, which makes the server overwhelmed easily in large scale applications. While in a P2P system, clients make requests to use resources on each other, so that the load can be evenly distributed on all participants in the network, making the system efficient and scalable.

2.2 Overlay Networks

P2P systems are usually constructed as overlays on top of existing physical networks in the application layer of the Internet OSI model. Currently, there are two general overlay construction methods for P2P networks: tree-based and swarm-based (also known as mesh-based, unstructured). The traditional method is to build a multicast tree on the overlay network. Christian in [29] introduces some algorithms for building multicast trees in IP multicast, such as spanning tree, reverse-path forwarding and center-based tree algorithms. Such algorithms can also be used in overlays. More complicated tree-based construction algorithms include [30–32]. The main advantage of a tree-based structure is, once a tree is built, data transmission can be simple and efficient: every node in the tree receives data from its parent and forward them to its children. But a tree topology has the following intrinsic defects:

- *Load balancing:* links near the source node have much higher traffic than those near leaf nodes.
- *Reliability:* a single link failure near the source node can lead to disconnection for many nodes.
- *Scalability:* whenever a node other than a leaf node leaves the multicast group, the tree must be reconstructed.
- *Complexity:* to construct an optimal multicast tree, we need to solve a Min-cost Steiner Tree problem, which is known to be NP-Hard.

All these defects of the tree-based structure have driven researchers to find more efficient ways to construct the overlay. In recent years, a swarm-based construction method has become more popular due to its simplicity and robustness [33–35]. In a swarm-based overlay, a peer joins the network by sending a lookup message to a tracker, which keeps track of all peers in the network. Then the tracker returns a list of potential neighbors according to some peer matching algorithms. The peer then can connect to other peers in the returned neighboring peer-list and exchange data with them. In this way, a highly resilient and decentralized overlay network can be constructed. The swarm-based overlay structure avoids many problems in the tree-based structure, with the cost of introducing some overheads on neighborhood management and data exchange for individual peers. Fortunately, these overheads are small in practice. The swarm-based method is now being adopted by many P2P systems, including Bittorrent [36], PPLive [25] and PPStream [26].

2.3 P2P Applications

Since the advent of the first popular P2P file sharing network, Napster [37], people have developed many P2P applications beyond the classical file sharing scheme. Here we list some popular P2P applications found in real systems.

2.3.1 P2P Computing

P2P technology can be used in distributed computing. In such a scenario, peers first sign up (usually voluntarily) for a computing project. A large and computationally hard problem is then decomposed into many small and independent problems by a centralized server, which are then distributed to all the peers in the network. Each peer carries out the computing tasks assigned to it, and once done, it sends the results back to the centralized server for further integration. There are two main benefits for this computing model. First, it utilizes the resources (CPU time, memory, etc.) on individual peers that would otherwise be wasted. Second, it is more robust than centralized computing, since individual peer failures can be recovered quickly, without affecting results computed on other peers. Such projects include SETI@Home [21], which is targeted to search for outer space intelligence and Einstein@home [22], which is used in searching for spinning neutron stars. Other similar projects can be found at BONIC [38].

2.3.2 P2P Storage

Traditional file storage systems use centralized access control to dedicated servers. P2P technology can be used to build distributed file systems that are more robust and have much larger storage space than traditional file storage systems. In such a system, data are stored on individual nodes of the P2P overlay network, and accessed by a global lookup scheme, such as Distributed Hash Table (DHT). A major difference between P2P file sharing systems and P2P storage systems is that P2P file sharing systems do not guarantee the availability and durability of the resources. While in a P2P storage system, resources are persistently stored in the network and can be accessed at any time. Examples of such applications are

Freenet [39], Cooperative File System (CFS) [23] and PATS [24]. Issues on designing a P2P storage system include redundancy management, access control, load balancing and fast resource location. Hasan et al. [40] give a survey on current P2P storage systems and analyze their advantages and disadvantages.

2.3.3 P2P Media Streaming

Media streaming is a special case of file sharing, in which peers consume data in real time when they download the media data from each other. The intrinsic real-time characteristic enforces more constrains on P2P media streaming systems than ordinary P2P file sharing systems. Since we focus on scheduling problems in P2P media streaming systems in this thesis, we discuss it in more details in the next section.

2.4 P2P Media Streaming Systems

The use of P2P technology to stream media contents, especially video programs, has gained much attention both in industry and in academia in recent years. Many P2P streaming systems have been deployed. In this section, we discuss some design alternatives and the main issues on designing an efficient P2P streaming system.

2.4.1 Live and On-demand Streaming

Current P2P media streaming systems can be categorized into two general types: live streaming and on-demand streaming, according to different types of services. In a P2P live streaming system, a program is usually created and streamed to clients synchronously. All peers (clients) watch the same part of the program at the same time. They can not pause and watch the same content in a later time, and they can not roll back to watch some previously missed parts. Such systems include CoolStreaming [9] and PULSE [41]. While in a P2P on-demand streaming system, media contents are usually pre-stored on media servers. Peers can join a streaming session at any time and start watching from any part of the program. They can also do operations like pause and roll back. Such systems include PPLive [25], PPStream [26], and UUsee [42]. The main difference between live streaming and on-demand streaming is that peers in live streaming systems are *more synchronized* than those in on-demand streaming systems, which can lead to different performance of the segment transmission scheduling algorithms as we will discuss later.

2.4.2 Tree-based and Swarm-bashed Structures

As we have stated before, P2P networks can be constructed in two different ways: tree-based and swarm-based. P2P streaming systems can be built on both structures. In view of the problems of tree-based structures as we have mentioned above, most of the recent P2P streaming systems, including PPLive and CoolStreaming, adopt swarm-based structure. We will use swarm-based structure to design and evaluate our proposed segment transmission scheduling algorithm in this thesis.

2.4.3 Push-based and Pull-based Protocols

Besides the different structures of P2P overlay network construction, there are two different ways of disseminating data in a P2P network: push-based protocol and pull-based protocol. In a push-based protocol, peers do not explicitly request data from others. After joining a streaming session, they just receive data from their upstream peers and forward them to their downstream peers in the network topology. Push-based protocol is usually used in conjunction with tree-based overlay structure, in which the peer in the root of the tree acts as a seeding peer, and pushes data down to the leaves of the tree, while other peers receive data from their parents and forward a copy of each received data to their children. This simple structure is efficient under the condition that the network topology does not change very often. While in practice P2P networks are quite dynamic. Peers join and leave a network session independently and frequently. This causes many P2P systems to resort to another data disseminating protocol: pull-based protocol. In a pull-based protocol, each peer explicitly requests data from its neighbors, and sends data to other neighbors that requesting data from it. In this way, peers *pull* data from others according to their own needs.

Compared to push-based protocol, the pull-based protocol incurs more overheads in the data requesting procedure, but it adapts much better to network dynamics. Whenever a neighbor fails or leaves, a peer can just redirect the requests to some other neighbors without affecting other peers in the network. Most of the swarm-based P2P networks use a pull-based protocol to exchange data among peers. Zhang et al. [14] provide a detailed analysis on the performance of the pull-based protocol in swarm-based P2P streaming systems. They also propose a hybrid push-pull protocol to improve the performance of the original pull-based protocol. Some other works also try to combine these two protocols to achieve the efficiency of push-based protocol and the flexibility of pull-based protocol, including [43] and [44]. In this thesis, we use pull-based protocol for exchanging data in our system.

2.4.4 Peer Matching

In a swarm-based P2P media streaming system with pull-based data dissemination protocol, each peer needs to find a list of neighbors that it can request data from. This process is called peer matching. Efficient peer matching can increase the efficiency of the system, and reduce the costs on Internet Service Providers (ISPs). Generally, an efficient peer matching algorithm tries to match peers that are close to each other to avoid long distance data transmission. A straightforward peer matching algorithm is random peer matching, in which each peer randomly selects a subset of peers in the network as its neighbors, and requests data from them. The random peer matching algorithm is used in some P2P systems for simplicity, but may not work well in practice. More intelligent peer matching algorithms are proposed in the literature in recent years. Bindal et al. [45] propose to select most neighbors within the same Anonymous System (AS) to reduce inter-ISP traffic. Choffnes et al. [46] suggest to use Content Delivery Networks (CDNs) in helping to find peers close to each others, and in [47], Hsu and Hefeeda propose two ISP-friendly peer matching algorithms that leverage publicly available information to infer the network topology for optimizing matching. Since in this thesis, we focus on the segment scheduling part rather than the peer matching part of the system, we use random peer matching in our system for simplicity.

2.4.5 Segment Transmission Scheduling

After a peer has found all the senders (neighbors), it has to decide from which senders to request which part of the data, in order to receive as many on-time data as possible in a P2P media streaming system. This is the scheduling problem that we have already introduced in the introduction chapter and we will analyze it in details in the following chapters.

2.5 Related Work

A measurement study on PPLive streaming system [25] reports that users suffer from long start-up delays and playout lags, and suggests that better segment scheduling algorithms are required [48]. Optimally computing segment schedules to maximize the perceived video quality, however, is computationally complex [9]. Therefore, many P2P streaming systems, such as [9, 10, 15, 49], resort to simple heuristics for segment scheduling. Pai et al. [10] propose to randomly schedule segment transmission, where each receiver randomly decides what segments to request from their neighbours. Zhang et al. [9] assume that segments with fewer potential senders are more likely to miss their deadlines, and propose to schedule the segments with fewer potential senders first. Agarwal and Rejaie [15] describe a weighted round-robin algorithm based on senders' bandwidths. In their algorithm, the number of segments assigned to each sender is in proportion to its bandwidth. In other words, senders with higher bandwidths will be assigned more segments than those with lower bandwidths. Kowalski and Hefeeda [49] propose to assign each segment to the sender that will deliver it the earliest. All these *heuristic* algorithms do not provide any performance guarantees on perceived video quality, and may not perform well in on-demand streaming systems [11].

One way to cope with the hardness of the segment scheduling problem is to *simplify* the objective function from the perceived video quality to the sum of *ad-hoc* utility functions [16, 17]. Zhang et al. [16] define a utility for each segment as a function of the rarity, which is the number of potential senders of this segment and the urgency, which is the time difference between the current time and the deadline of that segment. They then transform the segment scheduling problem into a min-cost flow problem. We note that although the min-cost flow problem can be optimally solved, the resulting schedules do *not* maximize the perceived video quality, which is the objective of the original problem.

Chakareski and Frossard [17] formulate an optimization problem to maximize the perceived video quality, and they solve it using an iterative descent algorithm. This algorithm, however, is computationally expensive and cannot be used in real-time systems. Therefore, they simplify the original formulation by proposing an ad-hoc utility function for each segment, which defines the multiplication of each segment's R-D (rate-distortion) efficiency, rarity, and urgency as its utility. They then greedily schedule the segments, i.e., they schedule the segments with higher utility values first. This greedy algorithm does *not* produce optimal schedules, nor does it provide any guaranteed performance. The works in [16,17] are different from our work in the sense that we solve the original segment scheduling problem, and we propose efficient approximation algorithm with guaranteed performance.

Several other works are related to the segment scheduling problem, but they do not directly solve it. Cai et al. [50] propose a P2P system that measures the time required to download the entire video from a number of senders and uses this information to choose senders from a large group of potential senders. Annapureddy et al. [11] propose using network coding to bypass the scheduling problem among small blocks belonging to the same relatively large segment by allowing all nodes to produce encoded data blocks. However, employing network coding may impose higher processing overhead on peers, which may require special hardware to speed up the decoding process [51], and is not easy to deploy. Several segment scheduling algorithms, such as [52], have been proposed for tree-based systems. They are, however, not applicable to swarm-based systems, in which peers have no knowledge on the global network topology.

Chapter 3

Segment Transmission Scheduling Problem

In this chapter, we first provide an overview of the P2P system model employed in this thesis. We then describe the segment transmission scheduling problem and show its hardness. For a quick reference, we list all symbols used in the thesis in Table 3.1.

3.1 System Model

We consider swarm-based P2P streaming systems, which are widely deployed currently over the Internet. Examples of such systems include CoolStreaming [9], PPLive [25], UUSee [42], SopCast [53], and TVAnts [54]. In these systems, peers form swarms for exchanging video data. Each swarm contains a subset of the peers, and a peer may participate in multiple swarms. Data availability on peers is propagated through exchanging control messages, such as buffer maps which indicate which video segments peers currently have in their buffers. Using these buffer maps, peers *pull* video segments from each other. More specifically, a receiving peer simultaneously requests segments from different sending peers. This is done by forming a segment transmission schedule by which the receiver specifies for each sender which segments to transmit and when to transmit. In video streaming systems, the arrival times of segments are critical, as segments arriving after their decoding deadlines cannot be rendered to users and are essentially useless.

The problem addressed in this thesis is to compute transmission schedules for receivers

Table 3.1: List of symbols used in the thesis.

Symbol	**Description**
δ	scheduling window size
F	frame rate
G	number of frames in each segment
M	number of senders
N	number of segments
T	number of time slots
m_i	sender i
n_i	segment i
t_i	transmission time for segment i
u_i	whether segment i arrives on-time
b_m	bandwidth of sender m
$\mathbf{Q}_m$	schedule for sender m
s_n	size of segment n
w_n	weight or value of segment n
d_n	deadline of segment n
$a_{n,m}$	availability of segment n on sender m
$x_{n,m,t}$	variable for m to transmit n at time t
α	perceived video quality
β	continuity index
γ	load balancing factor
z	total weight of ontime segments
$x_{n,m}$	assignment variable of segment n for sender m

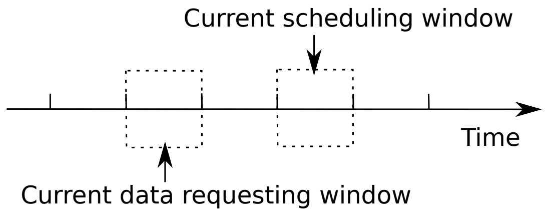


Figure 3.1: Two sliding windows in the system.

in order to optimize the number of on-time segments. Transmission schedules are computed for recurring sliding time windows, whose lengths are in the order of seconds. There are two sliding windows in the system: (i) the current data requesting window in which the receiver is requesting and receiving data from its senders, and (ii) the current scheduling window in which the scheduling algorithm is running to compute a transmission schedule, as illustrated in Fig. 3.1. The scheduling algorithm is running in a separate thread and is invoked several sliding windows ahead of the current data requesting window, so that when the data requesting and receiving thread receives all the data in the current sliding window and goes on to request data in the next sliding window, it does not need to wait for the scheduling results. But on the other hand, the current scheduling window should not be too far from the current data requesting window to avoid the situation that, at the time the scheduling algorithm is invoked, most of the segments are not available on its senders yet. As long as the scheduling algorithm runs faster than the data transmission speed, the client can continuously request data from its senders.

Our problem formulation and solution employ a realistic model for P2P streaming systems. Thus our proposed algorithm can readily be implemented in current swarm-based P2P streaming systems to improve their performance. Particularly, we assume that P2P streaming systems are highly dynamic and peers join and leave the systems frequently. Thus, we design our algorithm to be light-weight and can be invoked whenever such events occur in order to quickly recompute a new transmission schedule. In addition, the dynamic propagation and replication of video segments in the P2P streaming system can easily be handled by our algorithms. This is because segment propagation will trigger peers to update their buffer maps to reflect the availability of the newly acquired segments. When these buffer maps are exchanged among peers in control messages, our scheduling algorithms will account for the new segments in computing new transmission schedules.

It is important to emphasize that our work in this thesis focuses on a single, but critical, component of the P2P streaming system, which is the transmission scheduler. We present rigorous design of this component with mathematical formulation, complexity analysis, analytical guarantee on the performance of the computed schedules, and extensive simulations and experiments. We are *not* proposing a new, complete, P2P streaming system. However, we do not impose any assumptions on the other components of the system, e.g., the overlay management, peer matching, control messages exchange , churn handling, and incentive schemes. We assume that these components will function according to whatever protocols dictated by the specific P2P streaming system and eventually a set of potential senders will be presented to a receiver for requesting the video data. Given the data availability on each sender, our work is to make the best out of this set of senders for the receiver.

3.2 Problem Statement

We study the problem of transmitting a video stream from M senders to a receiver in a P2P streaming system. This stream consists of a series of coded video frames at frame rate F fps (frames per second), where each frame has a decoding deadline. Coded video frames that arrive at the receiver *after* their decoding deadlines are *useless*. To efficiently transmit video frames over the network, multiple consecutive coded frames are aggregated into a *segment*, which is the smallest transmission unit. Segment sizes are flexible and can be chosen based on the structure of the video stream. For example, one P2P streaming system may choose to construct a segment for each video frame for fine-grained scheduling, while another system may prefer to create a segment for every GOP (group-of-picture) for lower overhead. We consider a very general P2P streaming system that aggregates G coded frames in each segment, where video frames can have different sizes. We let N be the number of segments in the whole video stream. Since each segment consists of G coded frames, it has a playout time of G/F. Furthermore, segments are in different sizes because coded frames in video streams typically vary in sizes. We let s_n Kb be the size of segment n, where $1 \leq n \leq N$. Segment n has a decoding deadline $d_n = (n-1)G/F$ sec, which is the decoding deadline of the first video frame in that segment.

To generate feasible schedules, the receiver monitors its senders in terms of segment availability and uploading bandwidth. We let $a_{n,m}$ be the availability of segment n ($1 \leq n \leq N$) on sender m ($1 \leq m \leq M$). The receiver sets $a_{n,m} = 1$ if sender m has a copy of segment n, and $a_{n,m} = 0$ otherwise. We let b_m Kbps be the uploading bandwidth of sender m. With the segment availability and senders' bandwidth information, the receiver composes a segment schedule for a sliding window of δ sec, where δ is a system parameter, then it can request data according to the schedule, and senders transmit segments following the requests.

Our goal is to maximize the number of on-time segments. We exclude late segments as they cannot be used toward video quality improvement. With these notations, we formally describe the scheduling problem in the following.

Problem 1 (Segment Transmission Scheduling)**.** *We consider the segment transmission scheduling problem of video sequences in P2P streaming systems. The problem is to construct an optimal transmission schedule* $\mathbf{Q}_m = \{<m, n, t_n>,\ \ 1 \le n \le N_m\}$ *for each sender* m *in a scheduling window of* δ *sec, where* m *indicates the sender,* n *represents the segment,* t_n *is the starting transmission time of segment* n *on this sender, and* N_m *is the total number of segments scheduled to sender* m*. A segment* n *is said to be on-time if and only if* $t_n + s_n/b_m \le d_n$*. The objective is to maximize the number of segments that arrive on time at the receiver.*

3.3 Hardness

In the next theorem, we prove that solving the segment scheduling problem is computationally expensive.

Theorem 1 (Hardness)**.** *The segment transmission scheduling problem defined in Problem 1 is NP-Complete.*

Proof. We reduce the NP-Complete parallel machine scheduling problem [55, Sec. 5.3] to the segment scheduling problem. The machine scheduling problem schedules J jobs on K identical machines, and each job j $(1 \le j \le J)$ takes time p_j to complete. Each job can be scheduled on one machine and can not be preempted, and each machine can process one job at a time. The goal is to minimize the makespan $C_{\max}$, which is the maximal completion time of all jobs, and the problem is to determine whether there exists a schedule that $C_{\max} \le c_0$, for some given constant c_0.

For each machine scheduling problem, we construct a segment transmission scheduling problem as follows. Let the number of senders $M = K$, and the number of segments $N = J$. For each segment n $(1 \le n \le N)$, let $s_n = p_j$, and $d_n = c_0$. We let $b_m = 1$ for all $1 \le m \le M$, and $u_n = 1$ if segment n arrives on-time at the receiver from any of its senders, and $u_n = 0$ otherwise. Finally, the problem is to determine whether there exists a schedule that achieves $\sum_{n=1}^{N} u_n \ge N$. Since the segment scheduling problem is constructed and can be verified in polynomial time, and $C_{\max} \le c_0 \iff \sum_{n=1}^{N} u_n \ge N$, it is NP-Complete. □

Chapter 4

Proposed Approximation Algorithm

In this chapter, we first formally formulate the segment transmission scheduling problem, then introduce our approximation algorithm. A proof of its performance and complexity is then presented. We also design a load balancing mechanism for our scheduling algorithm at the end of this chapter.

4.1 Problem Formulation and Optimal Solution

In [56], Hsu and Hefeeda propose a formulation for the segment transmission scheduling problem. In their formulation, video segments are given different weights, which represent the relative importance of the segments for the rendered video quality. We call it the *weighted* version of the problem. They also present an approximation algorithm for the weighted segment transmission scheduling problem. The proposed algorithm in this thesis is totally different from the algorithm in [56]. First of all, our algorithm has an approximation factor of 2 while the algorithm in [56] has an approximation factor of 3. In addition, our algorithm is more computationally efficient, since it does not require solving any linear programming optimization problems. Furthermore, our algorithm optimizes the number of on-time segments and it does not need to compute the visual distortion in video frames contained in each segment, which is an expensive operation.

The problem we solve in this thesis can be considered as a special case of the problem proposed in [56], in which every segment is assigned a unit weight. We call it the *unweighted* segment transmission scheduling problem. We start by presenting the formulation for the weighted segment transmission scheduling problem. Then we present the formulation for the unweighted segment transmission scheduling problem considered in this thesis. In the evaluation chapters, we compare the performance of the proposed algorithm with the one in [56], and we show that both algorithms produce close results to the optimum and both outperform other algorithms proposed in the literature.

The weighted segment transmission scheduling problem can be formulated as a time-indexed ILP problem. Time-indexed formulations discretize the time axis into T time slots, where T is large so that the time slots are fine enough to represent any feasible schedule without reducing the value of the objective function. Let $x_{n,m,t}$ be a 0-1 variable for each $n = 1, 2, \ldots, N$, $m = 1, 2, \ldots, M$ and $t = 1, 2, \ldots, T$, where $x_{n,m,t} = 1$ if segment n is scheduled to be transmitted by sender m at time t, and $x_{n,m,t} = 0$ otherwise. Let w_n be the weight for segment n. Let $a_{n,m} = 1$ if sender m holds a copy of segment n, and $a_{n,m} = 0$ otherwise. Notice that, according to the definition, while the transmission of a segment often spans over several time slots, only the *first* time slot has an x value of 1. The weighted problem in [56] then can be formulated as:

$$z = \max \sum_{n=1}^{N} \sum_{m=1}^{M} \sum_{t=1}^{min\{d_n - \frac{s_n}{b_m}, T\}} w_n x_{n,m,t} \tag{4.1a}$$

$$\text{s.t.} \quad x_{n,m,t} \leq a_{n,m} \tag{4.1b}$$

$$\sum_{n=1}^{N} \sum_{j=t-\frac{s_i}{b_m}+1}^{t} x_{n,m,j} \leq 1 \tag{4.1c}$$

$$\sum_{m=1}^{M} \sum_{t=1}^{T} x_{n,m,t} \leq 1 \tag{4.1d}$$

$$x_{n,m,t} \in \{0,1\},\ a_{n,m} \in \{0,1\},$$
$$\forall\ n = 1, 2, \ldots, N,\ m = 1, 2, \ldots, M,\ t = 1, 2, \ldots, T. \tag{4.1e}$$

In this formulation, the objective function in Eq. (4.1a) is to maximize the sum of weights of all on-time segments, where the three summations iterate through all segments, senders, and time slots, respectively. Note that the last summation stops at time $d_n - s_n/b_m$ if

$d_n - s_n/b_m < T$, or T otherwise, because scheduling segment n *after* that time result in a late segment, which cannot improve video quality, and in each scheduling period we are only interested in scheduling segments within the scheduling window. The constraint in Eq. (4.1b) makes sure that we always schedule a segment to a sender who holds a copy of it, as it prevents the combination of $x_{n,m,t} = 1$ and $a_{n,m} = 0$ for all $t = 1, 2, \dots, T$. In Eq. (4.1c), observe that any segment n scheduled on sender m between time $t - s_n/b_m + 1$ and t would *occupy* the time slot t as transmitting segment n takes time s_n/b_m. Therefore, by considering all these segments, the constraint in Eq. (4.1c) ensures that at most one segment is scheduled on each sender at any time t. Last, the constraint in Eq. (4.1d) prevents segments from being scheduled on more than one sender.

The unweighted problem can be formulated in various ways, e.g., we can still use the time-indexed formulation as in Eq. (4.1). Just let $w_n = 1$ for all segments. However, doing so may prevent us from utilizing some unique properties of the unweighted scheduling problem. More specifically, we notice that the unweighted segment scheduling problem has the following property: Assuming N_m segments can be optimally scheduled to sender m (i.e., all of them arrive on time), scheduling these N_m segments in ascending order of their deadlines is one of the optimal schedules. We formally describe this property in the next theorem.

Theorem 2. *For N_m segments that can be scheduled on sender m, the schedule $\mathbf{Q}^*_m = \{<m, n, t_n>,\ \ 1 \le n \le N_m$ and $t_n = \sum_{i=1}^{n-1} s_i/b_m\}$ is an optimal schedule, where segments are sorted in ascending order on their deadlines.*

Proof. Let $\bar{\mathbf{Q}}_m$ be any optimal solution that maximizes the number of on-time arrival segments from sender m. We first eliminate any idle time in $\bar{\mathbf{Q}}_m$ by shifting all segments to their left whenever possible. This preprocessing step does not affect the optimality of $\bar{\mathbf{Q}}_m$ because all on-time segments complete no later than before. Next, if segments in $\bar{\mathbf{Q}}_m$ are sorted in ascending order on deadlines, letting $\mathbf{Q}^*_m = \bar{\mathbf{Q}}_m$ yields the theorem. Otherwise, we recursively apply the transformation described below.

Let $1 \le n_1, n_2 \le N_m$ be two arbitrary segments in $\bar{\mathbf{Q}}_m$, where n_1 is scheduled before n_2, but $d_{n_1} > d_{n_2}$. We update $\bar{\mathbf{Q}}_m$ using the following steps. First, we remove the segment n_1 from $\bar{\mathbf{Q}}_m$, which creates an idle time period. Second, we shift segments between n_1 (exclusive) and n_2 (inclusive) to their left for s_{n_1}/b_m sec. Third, we schedule n_1 immediately after the new location of n_2. Note that the new $\bar{\mathbf{Q}}_m$ is still optimal, because the shifted segments, except n_1, complete earlier, and the new n_1 still completes on time as $d_{n_1} > d_{n_2}$. We repeat these three steps until the segments in $\bar{\mathbf{Q}}_m$ are sorted, and we let $\mathbf{Q}^*_m = \bar{\mathbf{Q}}_m$. $\square$

This theorem shows that the segment assignment determines the optimality of segment schedules. This enables us to transform the segment transmission scheduling problem into an *assignment* problem, which is less complicated. Before presenting the formulation, we mention that similar theorems have been used to formulate machine scheduling problems, e.g., in [57, Sec. 3.3] and [58].

Theorem 2 states that sorting segments in ascending order on deadlines does not affect the existence of optimal schedules. Therefore, in the rest of this section, we assume $d_1 \leq d_2 \leq \cdots \leq d_N$. Otherwise, we sort and relabel the segments. We let $x_{n,m}$ ($1 \leq n \leq N$ and $1 \leq m \leq M$) be the *assignment variable*, where $x_{n,m} = 1$ if segment n is assigned to sender m, and $x_{n,m} = 0$ otherwise. We then formulate the problem as follows:

$$\begin{aligned} z = \quad & \max \sum_{n=1}^{N} \sum_{m=1}^{M} x_{n,m} && \text{(4.2a)} \\ \text{s.t.} \quad & x_{n,m} \leq a_{n,m} && \text{(4.2b)} \\ & \sum_{i=1}^{n} (s_i/b_m) x_{i,m} \leq d_n && \text{(4.2c)} \\ & \sum_{m=1}^{M} x_{n,m} \leq 1 && \text{(4.2d)} \end{aligned}$$

$$\begin{aligned} & x_{n,m} \in \{0,1\},\ a_{n,m} \in \{0,1\}, \\ & \forall\ n = 1, 2, \ldots, N \text{ and } m = 1, 2, \ldots, M. && \text{(4.2e)} \end{aligned}$$

In this formulation, by adding up all the assignment variables in Eq. (4.2a), the objective function is to maximize the number of on-time arrival segments. The first constraint in Eq. (4.2b) ensures that we always schedule a segment to a sender that holds a copy of it, as it prevents the combination of $x_{n,m} = 1$ and $a_{n,m} = 0$. The second constraint in Eq. (4.2c) computes the accumulated transmission time of sender m up to and including segment n, and checks whether segment n is completed before its deadline. In the third constraint in Eq. (4.2d), for each segment n, by adding up all its assignment variables on all senders and ensuring that it is less or equal to 1, it avoids assigning the segment to more than one sender. Notice that, compared to time-indexed formulation in Eq. (4.1), this formulation utilizes the unique property (Theorem 2) of the unweighted scheduling problem, and has fewer variables and constraints.

4.2 Overview of the Proposed Algorithm

For an optimal segment schedule, we can use any ILP solver to solve the optimization problem in Eq. (4.2). While this method may solve small scale segment scheduling problems, it cannot handle large problems in real time. Therefore, we present an efficient approximation algorithm in the following. The idea of our algorithm can be described as follows. For each scheduling period, the receiver takes all the segments in the current scheduling window and all its senders as input to the scheduling algorithm, and invokes it to compute a schedule for each sender sequentially. That is, the algorithm first tries to assign as many segments as possible to the first sender, then to the second sender and so on. On each sender, the algorithm repeatedly schedules the segment with the shortest transmission time first among all the segments that: (i) have not been scheduled to any senders and (ii) can be transmitted entirely by the current sender before their deadlines. The algorithm stops and returns the computed schedules for each sender when all the segments have been scheduled or when the bandwidths on all senders have been used up. The receiver then can request data from its senders according to the scheduling results, and the senders send data to the receiver according to the requests. Since the proposed algorithm sequentially schedules on senders, we call it Serialized Shortest Transmission-time First (SSTF) algorithm.

Fig. 4.1 presents a high-level pseudocode of the SSTF algorithm. This algorithm puts all segments in the set of $\bar{\mathbf{N}}$ in line 2, and sorts these segments in ascending order on segment size in line 3. Sorting segments allows us to efficiently locate the segment with the *shortest* transmission time from any sender m. This is because the bandwidth b_m is independent from which segment to send, and thus the transmission time of different segments on the same sender is proportional to the segment size. The algorithm then considers each sender sequentially in the for-loop from lines 4 to 11. The foreach-loop between lines 5 and 11 iterates through $\bar{\mathbf{N}}$ in ascending order on segment size, and the if-statement in line 7 identifies possible assignment by checking: (i) is segment n available on sender m, and (ii) can segment n be transmitted by sender m arrive on time? If both conditions hold, the algorithm schedules segment n on sender m by moving that segment

SSTF: Serialized Shortest Transmission-time First Algorithm

INPUTS:
(i) Segment sizes and deadlines in current scheduling window
(ii) Sender bandwidths and availability information
OUTPUT:
A schedule for each sender: $\mathbf{Q}_1, \mathbf{Q}_2, \ldots, \mathbf{Q}_M$

```
1.  let $\mathbf{Q}_m = \varnothing$, where $m = 1, 2, \ldots, M$
2.  let set $\bar{\mathbf{N}}$ consists of all remaining segments
3.  sort segments increasingly in $\bar{\mathbf{N}}$ on segment size
4.  for $m = 1$ to $M$ // sequentially considers sender $m$
5.    let $t = 0$ // consumed transmission time
6.    foreach segment $n \in \bar{\mathbf{N}}$ // from small to large
7.      if $a_{n,m} = 1$ and $t + s_n/b_m \leq d_n$
8.        // segment $n$ is available and can be transmitted on time
9.        add segment $n$ to $\mathbf{Q}_m$
10.       remove segment $n$ from $\bar{\mathbf{N}}$
11.       let $t = t + s_n/b_m$
12. return $\mathbf{Q}_1, \mathbf{Q}_2, \ldots, \mathbf{Q}_M$
```

Figure 4.1: The proposed approximation algorithm SSTF.

from $\bar{\mathbf{N}}$ to $\mathbf{Q}_m$. It also updates t, which represents the amount of time on sender m that has been consumed. Finally, the algorithm returns the segment transmission schedule in line 12.

4.3 Approximation Factor and Time Complexity

The proposed SSTF algorithm is an approximation algorithm with an approximation factor of 2 as will be shown in Theorem 3. We first present the formal definition of approximation factor in the following:

Definition Let Π be an optimization problem and let π be an instance of Π. Given an algorithm $\mathcal{A}$ for Π, let $\mathcal{A}(\pi)$ denote the value of the solution returned by $\mathcal{A}$ on instance π. Also, let $OPT(\pi)$ denote the optimal value for instance π.

Then, an approximation algorithm $\mathcal{A}$ for a maximization problem Π has an approximation factor of r if the following condition holds for all instances π of problem Π:

$$\frac{OPT(\pi)}{\mathcal{A}(\pi)} \leq r$$

Now, we analyze the approximation factor of the SSTF algorithm in the following theorem:

Theorem 3 (Approximation Factor)**.** *The SSTF algorithm given in Fig. 4.1 has an approximation factor of 2 for the segment transmission scheduling problem.*

Proof. We first consider a specific sender m in the for-loop between lines 4 and 11. We let $\mathbf{S_m}$ be the set of segments for schedule $\mathbf{Q_m}$ produced by the SSTF algorithm, and $\hat{\mathbf{S}}_\mathbf{m} = \bar{\mathbf{N}} \setminus \mathbf{S_m}$ be the set of segments for *any* schedule for sender m among the remaining segment list after the schedule of $\mathbf{Q_m}$. In addition, we use $|\mathbf{S_m}|$ and $\left|\hat{\mathbf{S}}_\mathbf{m}\right|$ to represent the number of segments in these two sets. We draw two observations. First, for any segment $\hat{s} \in \hat{\mathbf{S}}_\mathbf{m}$, there exists no segment $s \in \mathbf{S_m}$, such that the transmission time interval of $\hat{s}$ is a proper subset of that of s. Otherwise, $\hat{s}$ would be in $\mathbf{S_m}$ as the foreach-loop in lines 6–11 schedules the segment with the smallest segment size, which is equivalent to the shortest transmission time. Second, for any segment $\hat{s} \in \hat{\mathbf{S}}_\mathbf{m}$, there exists at least one segment $s \in \mathbf{S_m}$ such that the transmission time intervals of s and $\hat{s}$ overlap. Otherwise $\hat{s}$ would also be in $\mathbf{S_m}$ because of the foreach-loop. Combining these two observations, we have $\left|\hat{\mathbf{S}}_\mathbf{m}\right| \leq |\mathbf{S_m}|$.

Next, let $\mathbf{S} = \bigcup_{m=1}^{M} \mathbf{S}_m$ and $\mathbf{S}^* = \bigcup_{m=1}^{M} \mathbf{S}^*_m$, where $\mathbf{S}_m$ and $\mathbf{S}^*_m$ are the set of segments for schedules $\mathbf{Q_m}$ and $\mathbf{Q}^*_m$ of sender m determined by the SSTF and the OPT algorithms, respectively. We define $\mathbf{P} = \mathbf{S} \bigcap \mathbf{S}^*$ and $\mathbf{R} = \bigcup_{m=1}^{M} \left(\mathbf{S}^*_m \setminus \mathbf{S}\right)$. Therefore, we have $\mathbf{S}^* = \mathbf{P} \bigcup \mathbf{R}$ and $\mathbf{P} \subseteq \mathbf{S}$. Next, because $\mathbf{S}^*_m \setminus \mathbf{S}$ is a schedule for sender m, we have $|\mathbf{S}^*_m \setminus \mathbf{S}| \leq |\mathbf{S}_m|$ per the inequality developed in the previous paragraph. Since $\mathbf{S}_1, \mathbf{S}_2, \ldots \mathbf{S}_M$ are mutually disjoint and $\left(\mathbf{S}^*_1 \setminus \mathbf{S}\right), \left(\mathbf{S}^*_2 \setminus \mathbf{S}\right), \cdots \left(\mathbf{S}^*_M \setminus \mathbf{S}\right)$ are also mutually disjoint, we have $|\mathbf{R}| \leq |\mathbf{S}|$. Finally, we have $|\mathbf{S}^*| = |\mathbf{P} \bigcup \mathbf{R}| \leq |\mathbf{P}| + |\mathbf{R}| \leq |\mathbf{S}| + |\mathbf{S}| = 2\,|\mathbf{S}|$. □

Next, we show that the SSTF algorithm is efficient.

Theorem 4 (Time Complexity). *The SSTF algorithm given in Fig. 4.1 runs in time $O(MN + N \log N)$, where M is the number of senders and N is the number of segments.*

Proof. Sorting segments in line 3 takes time $O(N \log N)$. In addition, observe that the for-loop in lines 4–11 repeats for M times, and the foreach-loop in lines 6–11 iterates for up to N times. Thus, the time complexity of the SSTF algorithm is $O(M)O(N) + O(N \log N) = O(MN + N \log N)$. □

We notice that M and N are typically small values. This is because M is the number of potential senders for a given receiver, which is in the order of tens of senders, and N is the number of segments in each scheduling window, which is also in the order of a few tens of segments. For example, previous study [59] shows that there is a sweet range of M between 6 and 14, where the delivered quality to the majority of peers is high, and a typical value of N is 60 as used in CoolStreaming [9]. Thus, our algorithm can easily run in real time and can be invoked frequently to handle the high dynamics of P2P streaming systems.

4.4 Discussion

Although the SSTF algorithm achieves a small approximation factor, senders may be under diverse loads due to its serialized nature. More precisely, the SSTF algorithm may lead to unbalanced schedules, where senders $1, 2, \ldots, m-1$ ($1 \leq m \leq M$) are fully loaded while senders $m+1, m+2, \ldots, M$ are idle. Such unbalanced schedules are not desirable, because they discourage heavily-loaded senders from contributing, and underutilize the bandwidth of lightly-loaded senders. Fortunately, the load balancing issue only arises when *all* segments are scheduled by the SSTF algorithm; otherwise, the bandwidths of all senders are used up, and the loads are already balanced.

To address the potential load balancing issue, we modify the SSTF algorithm as follows: We search for a maximum transmission time c of all senders, which is long enough to transmit all segments on time, yet short enough to prevent idle time on senders. We initialize $c = \delta$. Then we add another condition $t + s_n/b_m \leq c$ to line 7 in Fig. 4.1, which limits the total transmission time, and thus the load on each sender. We run this modified SSTF algorithm several times, and search for the optimal transmission time c^* using binary search. More specifically, we start from interval $[c_l = 0, c^* = \delta]$. We then let $c = (c_l + c^*)/2$ and run the SSTF algorithm with the new c value to check whether all segments can be scheduled on time. We update the interval with $[c, c^*]$ if not all segments are on time, otherwise we update it with $[c_l, c]$. We stop the search once reaching the maximum number of iterations I, and we return the segment schedule corresponding to the current c^*. Since the typical scheduling window is short, the binary search usually stops after very few rounds. We set a conservative value of I to be 8 in our implementation. Since I is a small constant, this modification can be applied to the SSTF algorithm, which has a very low time complexity as shown in Theorem 4.

Chapter 5

Evaluation Using Simulation

In this chapter, we first describe the setup of our simulation and define several performance metrics used in the evaluation. We then present the evaluation results of the SSTF algorithm, and compare them with the results of some existing scheduling algorithms.

5.1 Simulation Setup

We have implemented an event-driven simulator in Java to evaluate the performance of the proposed segment scheduling algorithm. Five scheduling algorithms are implemented in this simulator: RF [9], MC [16], SSTF, WSS [56] and OPT [56]. The RF algorithm implements the rarest first algorithm. It schedules the segment with the fewest potential senders first, and among multiple senders, the one with highest bandwidth and enough available time first. RF is a fairly common algorithm used in several widely deployed P2P systems, such as CoolStreaming [9] and PPlive [25]. MC is a quite sophisticated algorithm that has recently been proposed in the literature.

The MC [16] algorithm is based on an ILP formulation, which is converted into a min-cost flow problem and solved by combinatorial algorithms. While we employ the same utility function defined in the evaluation section of [16], the original algorithm can only schedule transmission of fixed-size blocks. We, therefore, extend that algorithm to support variable-size segments by: (i) dividing each segment into blocks, (ii) solving the block transmission problem using their algorithm, and (iii) for each segment, we try to schedule it on the sender which has been assigned the most number of blocks. If that sender has used up all the bandwidth, we then try to schedule it on the sender which has been assigned the second

Table 5.1: List of video parameters used in the thesis.

Video Name	**SonyDemo**	**Terminator 2**
Resolution	CIF 352*288	HD 1280*720p
Frame Rate (fps)	30	30
Number of Frames	9012	9010
Group of Pictures (GOP)	16	12
Quantization Parameter (QP)	16	28
Frame Size (bits): min / max / mean	136 / 524416 / 73260	592 / 530664 / 85048
PSNR (dB): min / max / mean	41.9 / 49.1 / 45.0	36.0 / 54.1 / 40.7
Mean Bitrate (Kbps)	2200.8	2554.3

Table 5.2: Peer uploading bandwidth distribution.

Distribution (%)	10.0	14.3	8.6	12.5	2.2	1.4	6.6	28.1	16.3
Total Bandwidth (Kbps)	256	320	384	448	512	640	768	1024	> 1500
Contributed Bandwidth (Kbps)	150	250	300	350	400	500	600	800	1000

most number of blocks and so on.

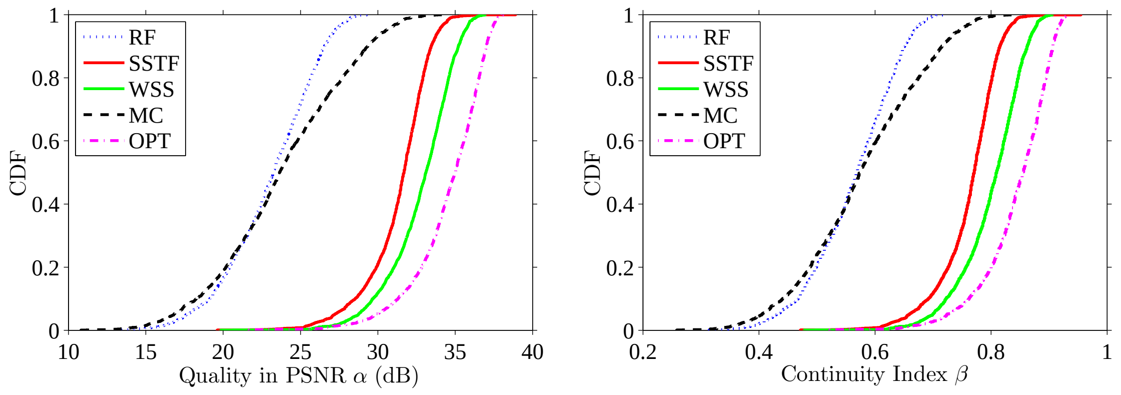
SSTF is the implementation of our approximation algorithm and WSS is the implementation of the algorithm proposed in [56] . In the OPT algorithm, we directly solve the ILP formulation 4.1 offline with an ILP solver CPLEX. In particular, the CPLEX package provides a set of Java class libraries that allow us to specify and solve our problems using Java syntax through JNI (Java native interface).

We choose two high quality videos with different characteristics from the Arizona State University video trace library [60] in the simulation to analyze the performance of our algorithm. Table 5.1 summarizes the main parameters of the videos. We use PSNR (Peak Signal-to-Noise Ratio) as the perceived video quality metric and as the segment weight (w_n) in our simulation. The simulator puts every GOP (Group-of-Picture) into one segment, so that segment n has a decoding deadline of $G(n-1)/F$, where $n = 1, 2, \ldots, N$. Packing each GOP into a scheduling unit ensures that every received segment can be decoded independently at the receiver, since dependency among video frames is restricted within a GOP. Some previous algorithms, like the original MC algorithm proposed in [16], divide videos into fixed size blocks as scheduling units. This can cause segments not to be decodable, because even if a very small part of a segment is not received on time, the whole segment cannot be decoded, which results in degraded video quality.

We simulate a dynamic system with a total number of 2,000 peers. We initially pre-deploy the video sequences on only 1% of the peers chosen randomly, which forms the initial seeding peers. We run the simulation for 24 hours for each algorithm. Individual peers dynamically join and leave a swarm at different times during the streaming of a video sequence. The joining and leaving times are randomly chosen from the whole simulation time period following a uniform distribution. Upon joining a swarm, each peer is instructed to stream the video sequence. We also simulate dynamic replication of video segments as peers can upload segments as soon as they have downloaded those segments from other peers. We consider a peer matching service that randomly provides each new peer up to 10 senders. Each new peer then connects to these senders, runs the scheduling algorithm, and requests segments following the scheduling results. We set the scheduling window to be 10 sec in the simulation.

The simulator determines each sender's uploading bandwidth following the distribution given in Table 5.2. This bandwidth distribution is proposed in a recent paper [61] based on various measurement studies on both corporate and residential users. We note that peers would *not* contribute all their bandwidth to P2P streaming, because doing so would slow down other Internet applications such as email and Web. The *contributed bandwidth* of each class of peers is also given in this table as recommended in [61]. With the randomly chosen bandwidth, the simulator fairly distributes available bandwidth among all connections, and predicts the transmission time for each packet accordingly. Finally, in the OPT and WSS algorithms, the system parameter T is set to be 100, which means the time slots are 100 msec long in our formulation.

We run the simulator independently for each considered algorithm on a commodity PC, with a 2.66GHz Quad-Core Intel Xeon CPU and 8GB memory. We define three performance metrics: the average perceived video quality α, the continuity index β, and the load balancing factor γ. The average perceived video quality is computed by assuming that all late segments result in zero PSNR, and defining $\alpha = \sum_{n=1}^{N} w_n u_n / N$, where w_n is the average perceived video quality of video frames in segment n. In our setup, it is the average PSNR value of the segment. We define the continuity index as the number of segments that arrive by their decoding deadlines over the total the number of segments in the video.



(a) Video quality (b) Continuity index

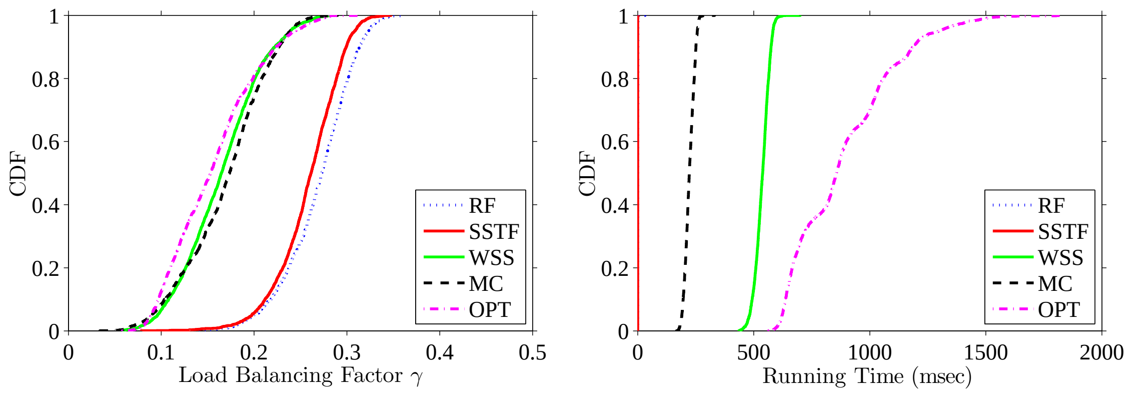
Figure 5.1: Overall comparison of the proposed (SSTF) algorithm and the Rarest First (RF) algorithm used in several deployed systems and the Min-Cost flow based (MC) algorithm and the Weighted Segment Scheduling (WSS) algorithm recently proposed in the literature and the optimal algorithm (OPT) solved directly from the ILP formulation. (Terminator 2 video)

That is, $\beta = \sum_{n=1}^{N} u_n/N$. Last, we define the load of sender m as its uploading bandwidth utilization, which is $\left(\sum_{n\in\mathbf{Q}_m} s_n/\delta\right)\Big/ b_m$, where $\sum_{n\in\mathbf{Q}_m} s_n$ accounts for the total size of all segments scheduled on that sender in one scheduling period and δ is the length of scheduling window. The load balancing factor γ is then computed as the standard deviation of loads for all scheduling periods on that sender. Similar performance metrics are used in other works in the literature, such as [9, 62].

5.2 Simulation Results

Overall Comparison. For each simulation, we calculate the average performance for each peer across all the scheduling periods. We iterate through all peers and compute the CDF (Cumulative Distribution Function) curves of each performance metric. We repeat the same computation for each scheduling algorithm. The results are summarized in Fig. 5.1.

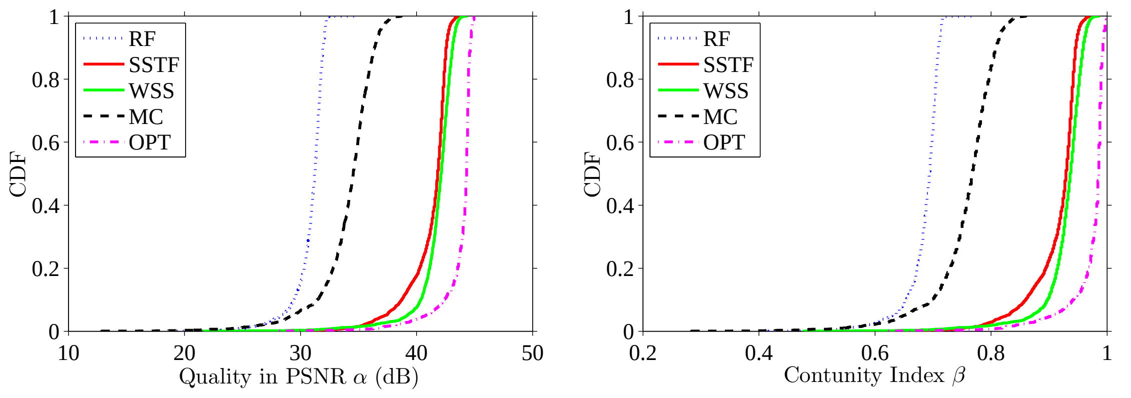
In the first simulation using video Terminator 2, we plot the video quality in Fig. 5.1(a). This figure shows that the WSS and SSTF algorithms substantially outperform the other two heuristic algorithms RF and MC, and they stay very close to the OPT. First, for the lowest 5% of the peers in terms of perceived video quality, the WSS and SSTF algorithms



(a) Load balance

(b) Running time

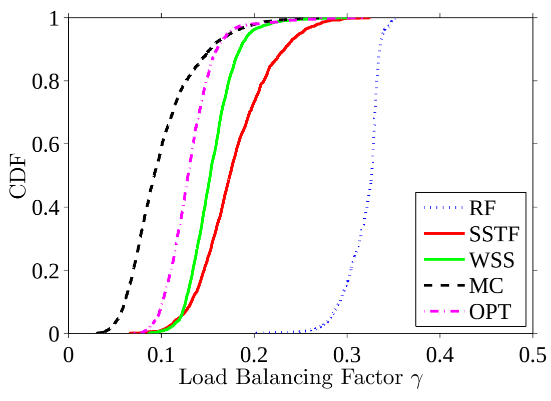
Figure 5.2: Load balance and running time of the WSS, SSTF, RF, MC and OPT algorithms. Note: the running time curves for the SSTF and RF algorithms are not visible in the figure because they are very close to 0. (Terminator 2 video)



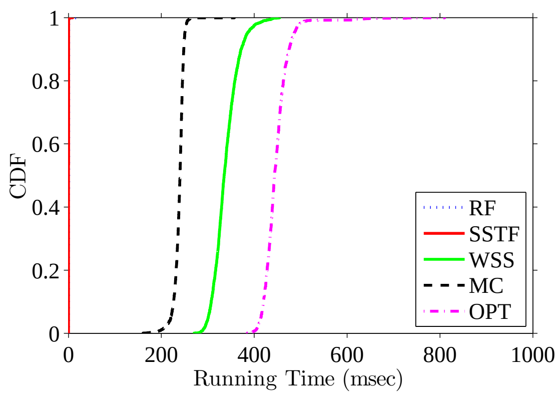
(a) Video quality

(b) Continuity index

Figure 5.3: Overall comparison of the proposed (SSTF) algorithm and the Rarest First (RF) algorithm used in several deployed systems and the Min-Cost flow based (MC) algorithm and the Weighted Segment Scheduling (WSS) algorithm recently proposed in the literature and the optimal algorithm (OPT) solved directly from the ILP formulation. (SonyDemo video)



(a) Load balance



(b) Running time

Figure 5.4: Load balance and running time of the WSS, SSTF, RF, MC and OPT algorithms. Note: the running time curves for the SSTF and RF algorithms are not visible in the figure because they are very close to 0. (SonyDemo video)

achieves an average perceived video quality of at least 27 dB and 28 dB, respectively, where the optimal solution achieves 29 dB, while the MC and RF algorithm can only achieve 16 dB and 17 dB at that level. Second, both the WSS and SSTF algorithms achieve more uniform and higher video quality for all peers compared to the other two heuristic algorithms. This is shown by the concentration of the CDF curves between 25 dB and 35 dB by the two algorithms, while the CDF curves of the other two heuristic algorithms are spread over larger ranges, from 15 dB to up to 30 dB.

This simulation results show that WSS performs a little better than SSTF, but notice that the computational complexity of the WSS algorithm is much larger than that of the SSTF algorithm. WSS works very well on fast machines with enough computing power, but may not perform well on slow machines with limited computational resources. We will see this point in the next chapter.

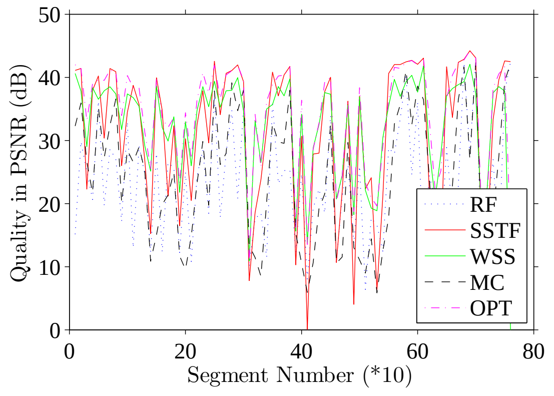
We then report the continuity index in Fig. 5.1(b). This figure illustrates that the WSS and SSTF algorithm results in much higher continuity index than the RF and MC algorithms. For example, more than 98% of the peers observe a continuity index of at least 60% using the SSTF and WSS algorithm, while less than 40% of the peers observe that continuity index for the RF and MC algorithms. This means employing the WSS and SSTF algorithms significantly reduces the playout glitches at receivers. We also notice that the results of the unweighted video quality metric (continuity index) is very similar to the results of the weighted video quality metric. This indicates that our previous assumption that all segments have same weight is reasonable in practice. This is because we stream single layered videos in our model and the weights of segments are close to each other in single layered videos. Fig. 5.1 clearly shows that the proposed SSTF algorithms results in much higher and smoother video streaming quality, compared to the MC algorithm and the widely deployed RF algorithm, and performs very close to the WSS algorithm, but much faster.

We next plot the load balancing factor in Fig. 5.2(a). Excessive load balancing factor may slow down some senders, which could discourage users from contributing to the P2P network. This figure illustrates that the WSS algorithm achieves at most 29% deviation, and thus distributes the transmission load fairly among senders; The SSTF algorithm occurs a little bit higher load balancing factor, but still within 35% deviation, and at the same time they produce better video quality and higher continuity index as shown in Figs. 5.1(a) and 5.1(b). While the other two algorithms result in similar diversity in the load imposed on peers but worse video quality as shown in the figures.

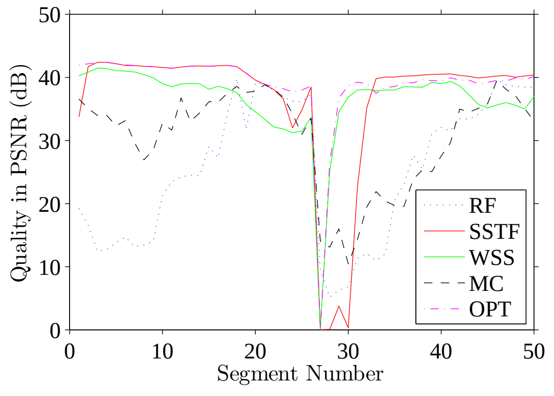
Last, we plot the average running time across all the scheduling periods on each sender in Fig. 5.2(b). We can see that the SSTF and RF algorithms run much faster than the others. Notice that the running time curves for the SSTF and RF algorithms are not visible in the figure because they are very close to 0. On the other hand, except for the OPT algorithm, all the other algorithms can run in the scale of milliseconds, which is small compared to the scheduling window. For the MC algorithm, although the min-cost flow problem can be solved in polynomial time, converting the ILP problems to min-cost flow problems as proposed in [16] can result in large number of nodes in the model, thus it runs much slower than the SSTF and RF algorithms. Notice that for the OPT algorithm, we set a bound on the maximum number of iterations for the ILP solver to prevent it from taking too long to solving the ILP problem, and we omitted such cases in the figure. In summary, the figure shows that our proposed algorithm can easily run in real time. In contrast, computing a schedule using the OPT algorithm may take a long time, especially for those slow machines.

Figs. 5.3 and Figs. 5.4 show the results for the second simulation using video SonyDemo. We can observe that using videos with different characteristics, the overall results are not changed much.

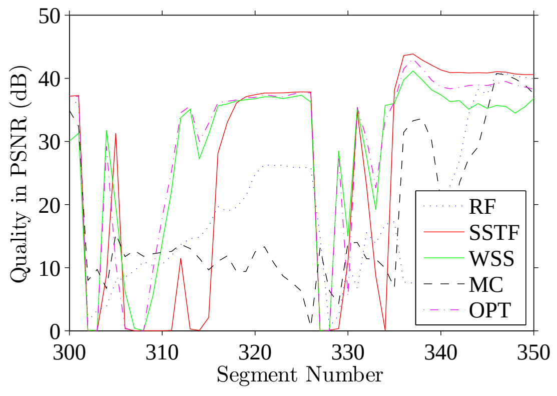
Segment level Comparison. Next we iterate through all the video frames, and compute the average PSNR in each segment for every receiving peer. We then compute the average video quality across all peers. We repeat the computation for all scheduling algorithms. To clearly show the results of the whole video sequence, we aggregate every 10 segments and compute their average PSNR. Results are shown in Fig. 5.5(a) for the first simulation (with video Terminator 2). The figure shows that the WSS and SSTF algorithms perform much better than the other two heuristic algorithms throughout the whole video period. We notice that there are a number of short-period drops in the whole sequence. These are caused by the missed or late segments. The figure shows that the number of missed segments (quality drops) is much smaller in SSTF and WSS compared to RF and WC. To clarify it further, we zoom in several regions of the Fig. 5.5(a) to individual segment level, and extract three intervals from the whole sequence: The first 50 segments, 50 segments from the middle, and 50 segments from the last part of the video, and plot them in Fig. 5.5(b), Fig. 5.5(c) and Fig. 5.5(d), respectively. These figures clearly show that the SSTF and WSS algorithms result in higher perceived video quality. The RF and MC algorithms lead to too many quality drops, which degrade user experience. In summary, Fig. 5.5 confirms that employing the WSS and SSTF algorithms yield higher perceived video quality and fewer playout glitches, compared to the RF and MC algorithms.



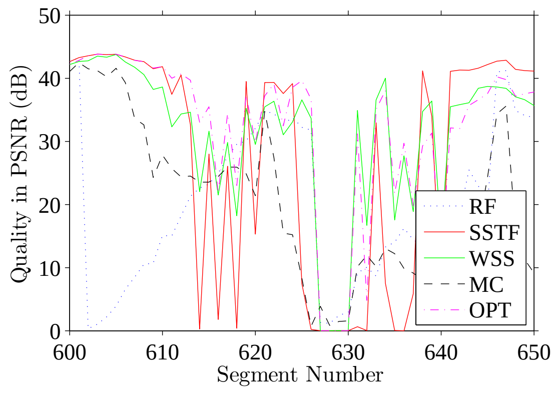
(a) Quality on the whole video sequence



(b) Quality sample (i)



(c) Quality sample (ii)



(d) Quality sample (iii)

Figure 5.5: Segment level comparison of the proposed (SSTF) algorithm and the Rarest First (RF) algorithm used in several deployed systems and the Min-Cost flow based (MC) algorithm and the Weighted Segment Scheduling (WSS) algorithm proposed in the literature and the optimal algorithm (OPT) solved directly from the ILP formulation. (Terminator 2 video)

Chapter 6

Evaluation Using Prototype Implementation

In order to evaluate the proposed algorithm in a real system, we have developed a prototype P2P streaming system and deployed it on PlanetLab [19]. We have also implemented the WSS, RF and MC algorithms in the prototype. In this chapter, we first describe our experimental setup and then present our results.

6.1 Experimental Setup

Fig. 6.1 shows the high level diagram of our prototype system implementation. The system consists of one tracker and a set of peers interested in streaming a video file (only two peers are shown in the figure). The tracker is used to coordinate all peers. It keeps a list of all currently active peers in the peer manager module and does peer matching in the peer matching module when a peer asks for a list of neighbours. There are several peer matching algorithms proposed in the literature, such as [47] and [46]. Since in this thesis we mainly focus on the scheduling part of the system, we use a random peer matching algorithm just for its simplicity. That is, the tracker randomly selects a list of peers in the peer list and returns them to the requesting peer.

On the peer side, each peer contains three main modules: the peer manager module that establishes and maintains connections with other peers; the buffer map manager module that keeps track of the availability of segments on this peer; and the scheduler module that does

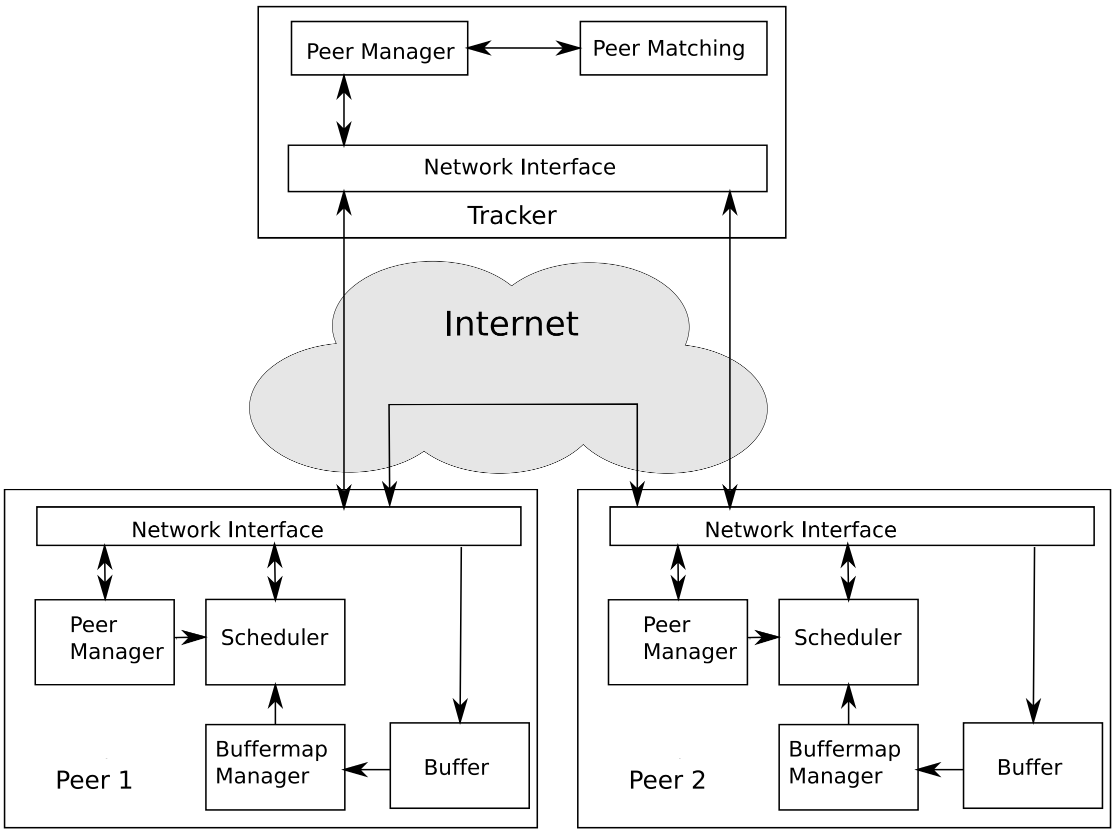


Figure 6.1: A high level diagram for the prototype P2P streaming system implementation.

segment transmission scheduling based on its neighbourhood information and buffer map availability information.

The system works as follows:

1. The tracker starts by listening on a port that is known to all the peers.

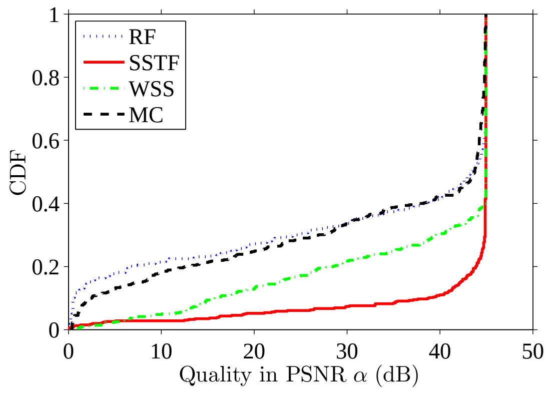
2. All peers start by first connecting to the tracker in time randomly distributed across the whole experiment period. The first small part of the peers connected to the tracker will be set to the seeding peers, which have the whole video sequence. The rest of peers get a list of neighbours once they have connected to the tracker according to the peer matching algorithm, and start to exchange data with their neighbours.

3. On each peer, for each scheduling period, the scheduler computes a segment transmission schedule based on the senders and segments information and then the peer requests data from its senders accordingly. The received data is put into the buffer for playout and the buffer map is then updated according to the received data for the next scheduling period. This process continues until the end of the streaming session.

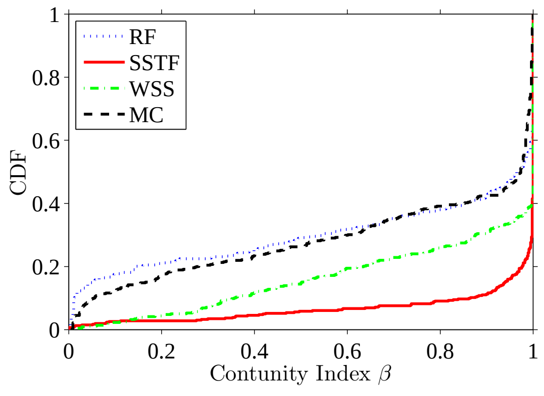
4. For buffer map exchange, we use a periodic broadcasting scheme: each peer periodically checks its own buffer map (in our setup, every 5 sec). If the buffer map is updated since last time checked, the peer broadcasts the updates to all its receiving peers. In this way, the buffer map availability information is exchanged between the peers.

5. The experiment stops when all peers have reached the end of the streaming session or when the experiment period expires.

The experiments involved 500 PlanetLab nodes distributed across the world. We stream the SonyDemo video that we used in the simulation. We initially pre-deploy the video on 5% of the nodes, and let other nodes join and leave the P2P network at time randomly distributed during the whole experiment period. We set the number of senders to 10 and use a random peer matching algorithm to find senders for each receiver. We use a scheduling window of 5 sec, and an initial delay of 2 sec for all algorithms. These are typical system parameters as suggested in previous works [9,16,59]. The current sending rate of a sender is estimated from its previous sending rates in a moving window manner, which is a commonly used technique for bandwidth estimation.

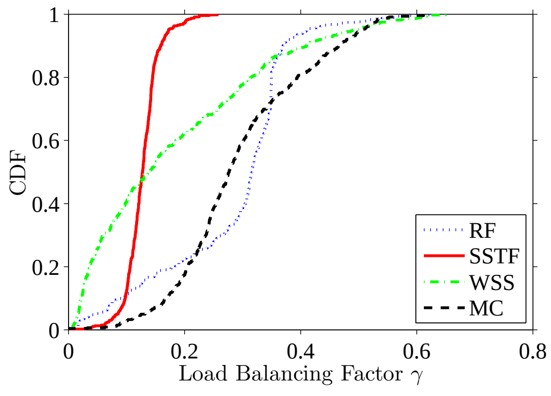


(a) Video quality

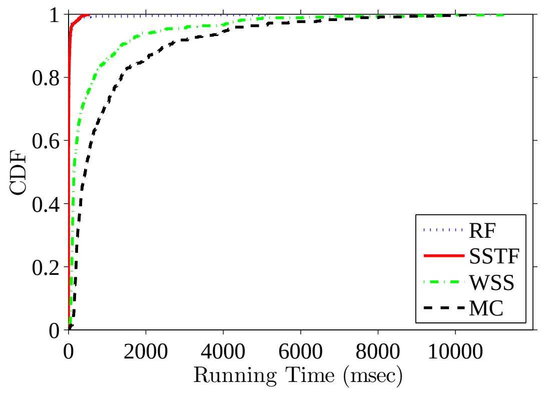


(b) Continuity index

Figure 6.2: Overall comparison in PlanetLab-based experiments. (SonyDemo video)



(a) Load balance



(b) Running time

Figure 6.3: Load balance and running time in PlanetLab-based experiments. (SonyDemo video)

6.2 Experimental Results

We first plot the video quality in Fig. 6.2(a). This figure shows that the SSTF algorithm achieve better quality than the other three. For example, at the quality level of 30 dB, there are approximately 90%, 75%, 65% and 65% peers achieved that level, for the SSTF, WSS, RF and MC algorithms, respectively. Another thing we notice is that, compared to the results from simulation, the SSTF algorithm performs better than the WSS algorithm. This is because some of the nodes on PlanetLab are quite busy, their CPU times are shared by many other users. As we have stated before, the WSS algorithm involves solving a set of linear programming problems, which usually requires considerable computation. The scheduling algorithm runs several scheduling windows ahead of current data requesting window, and if the algorithm runs slower than the data transmission speed, the receiver have to wait for the scheduling results, which will degrade the performance of the algorithm. During the experiment, we observed that such situation happens occasionally on some of the slow machines on PlanetLab. In summary, SSTF performs very close to WSS on faster machines, and much better than WSS on slow machines, as we can see from the results of both simulation and experiment.

We then report the continuity index in Fig. 6.2(b). This figure illustrates that the SSTF algorithm results in much higher continuity index than others: more that 90% of the peers can achieve a continuity index of at least 80%. The WSS algorithm comes next: more than 74% of the peers achieve the same continuity level. While the other two lead to poor continuity: approximately 60% for both RF and MC, at the continuity level of 80%. We next plot the load balancing factor in Fig. 6.3(a). This figure illustrates that the SSTF algorithm distributes the transmission load across senders in a fairer manner than the other three algorithms. Last, we we plot the average running time in Fig. 6.3(b). It shows that the SSTF algorithm runs faster than any of the other algorithms, thus can be used on slow machines that have limited computational resources.

In summary, both the results from simulations and experiments show that our proposed SSTF algorithm can achieve better video quality and more balanced load compared to other widely used algorithms for the segment scheduling problem in P2P streaming systems.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

We studied the segment transmission scheduling problem in P2P video streaming systems, where a receiver periodically computes a transmission schedule for all its senders to maximize the number of on-time segments. We consider both live and on-demand P2P streaming systems. We proved that this problem is NP-Complete. We formulated the considered problem with an Integer Linear Programming (ILP) formulation. Optimally solving this ILP problem, however may take prohibitively long time, and is not suitable for P2P video streaming systems. Thus we proposed an efficient approximation algorithm and formally showed that it provides an approximation factor of 2 in the worst case. To the best of our knowledge, this is the best approximation factor achieved so far for the segment transmission scheduling problem in P2P streaming systems.

We implemented an event-driven simulator as well as a P2P prototype running on PlanetLab, and conducted extensive simulations and experiments to evaluate the proposed algorithm. The results showed that the proposed algorithm achieves much higher perceived video quality and continuity index and fairer load balancing across senders compared to other algorithms used in current systems.

In summary, the proposed algorithm not only provides analytical guarantees on the worst-case performance, but it also has superior average-case performance in practice compared to other scheduling algorithms proposed in the literature and used in the deployed P2P streaming systems. Furthermore, our algorithm is computationally efficient and thus can be implemented in actual P2P streaming systems for both live and on-demand services.

7.2 Future Work

This work can be extended in several directions. One possible extension is to implement the scheduling algorithms as plugins for some existing P2P systems, such as Vuze [63], and get some results from these already-deployed systems. Furthermore, our current algorithm only works for single-layered videos. Another possible extension is to design scheduling algorithms for scalable video streams with guaranteed performance. In this case, the scheduling problem is more complicated, since there is no easy way to pack frames from different layers into segments so as to achieve both efficiency and scalability. Furthermore, frames in scalable video streams have different degrees of importance to the perceived quality of the video, i.e., frames from the base layer are more important than those from the enhancement layer. More intelligent scheduling algorithms are needed to tackle these problems. Finally, in our prototype system, we only implemented simple algorithms for the other parts of the systems, such as the random peer matching algorithm. It will be interesting to study the interaction of the proposed algorithm with other parts of the P2P streaming system, such as better peer matching algorithms.

Bibliography

[1] B. Liu, Y. Cui, B. Chang, B. Gotow, and Y. Xue. BitTube: case study of a web-based peer-assisted video-on-demand system. In *Proc. of IEEE International Symposium on Multimedia (ISM'08)*, pages 242–249, Berkeley, CA, December 2008.

[2] P. Rodriguez, S. Tan, and C. Gkantsidis. On the feasibility of commercial, legal P2P content distribution. *ACM SIGCOMM Computer Communication Review (CCR'06)*, 36(1):75–78, January 2006.

[3] Y. Tu, J. Sun, M. Hefeeda, and S. Prabhakar. An analytical study of peer-to-peer media streaming systems. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 1(4):354–376, November 2005.

[4] D. Xu, S. Kulkarni, C. Rosenberg, and H. Chai. Analysis of a CDN-P2P hybrid architecture for cost-effective streaming media distribution. *ACM/Springer Multimedia Systems Journal*, 11(4):383–399, April 2006.

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

[6] Y. Chu, S. Rao, and H. Zhang. A case for end system multicast. In *Proc. of ACM International Conference on Measurements and Modeling of Computer Systems (SIGMETRICS'00)*, pages 1–12, Santa Clara, CA, June 2000.

[7] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: High-bandwidth multicast in cooperative environments. In *Proc. of ACM Symposium on Operating Systems Principles (SOSP'03)*, pages 298–313, Bolton Landing, NY, October 2003.

[8] V. Padmanabhan, H. Wang, P. Chou, and K. Sripanidkulchai. Distributing streaming media content using cooperative networking. In *Proc. of ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'02)*, pages 177–186, Miami Beach, FL, May 2002.

[9] X. Zhang, J. Liu, B. Li, and T. Yum. CoolStreaming/DONet: A data-driven overlay network for peer-to-peer live media streaming. In *Proc. of IEEE INFOCOM'05*, pages 2102–2111, Miami, FL, March 2005.

[10] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. Mohr. Chainsaw: Eliminating trees from overlay multicast. In *Proc. of International Workshop on Peer-to-Peer Systems (IPTPS'05)*, pages 127–140, Ithaca, NY, February 2005.

[11] S. Annapureddy, S. Guha, C. Gkantsidis, D. Gunawardena, and P. Rodriguez. Is high-quality VoD feasible using P2P swarming? In *Proc. of International World Wide Web Conference (WWW'07)*, pages 903–912, Banff, Canada, May 2007.

[12] N. Magharei, R. Rejaie, and Y. Guo. Mesh or multiple-tree: A comparative study of live P2P streaming approaches. In *Proc. of IEEE INFOCOM'07*, pages 1424–1432, Anchorage, AK, May 2007.

[13] C. Liang, Y. Guo, and Y. Liu. Is random scheduling sufficient in P2P video streaming? In *Proc. of IEEE International Conference on Distributed Computing Systems (ICDCS'08)*, pages 53–60, Beijing, China, June 2008.

[14] M. Zhang, Q. Zhang, L. Sun, and S. Yang. Understanding the power of pull-based streaming protocol: Can we do better? *IEEE Journal on Selected Area in Communications*, 25(9):1678–1694, December 2007.

[15] V. Agarwal and R. Rejaie. Adaptive multi-source streaming in heterogeneous peer-to-peer networks. In *Proc. of SPIE/ACM Multimedia Computing and Networking (MMCN'05)*, pages 13–25, San Jose, CA, January 2005.

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

[17] J. Chakareski and P. Frossard. Utility-based packet scheduling in P2P mesh-based multicast. In *Proc. of SPIE International Conference on Visual Communication and Image Processing (VCIP'09)*, page 72571S, San Jose, CA, January 2009.

[18] Y. Shen, C. Hsu, and M. Hefeeda. Efficient algorithms for multi-sender data transmission in swarm-based peer-to-peer streaming systems. *Submitted to IEEE Transactions on Multimedia*, July 2010.

[19] PlanetLab Home Page. `http://www.planet-lab.org/`.

[20] BitTorrent Home Page. `http://www.bittorrent.com/`.

[21] SETI@home Home Page. `http://setiathome.ssl.berkeley.edu/`.

[22] Einstein@home Home Page. `http://einstein.phys.uwm.edu/`.

[23] F. Dabek, F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proc. of ACM symposium on Operating Systems Principles (SOSP'01)*, pages 202–215, Banff, Canada, October 2001.

[24] A. Rowstron and P. Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. *ACM SIGOPS Operating Systems Review*, 35(5):188–201, October 2001.

[25] PPLive Home Page. `http://www.pplive.com/`.

[26] PPStream Home Page. `http://www.ppstream.com/`.

[27] Facebook Home Page. `http://www.facebook.com/`.

[28] eBay Home Page. `http://www.ebay.com/`.

[29] H. Christian. *Routing in the Internet (Second Edition)*. Prentice Hall PTR, 1999.

[30] B. Zhang, S. Jamin, and L. Zhang. Host multicast: A framework for delivering multicast to end users. In *Proc. of IEEE INFOCOM'02*, pages 1366–1375, New York, NY, June 2002.

[31] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. *ACM SIGCOMM Computer Communication Review*, 32(4):205–217, October 2002.

[32] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: high-bandwidth multicast in cooperative environments. In *Proc. of ACM symposium on Operating systems principles (SOSP'03)*, pages 298–313, Bolton Landing, NY, October 2003.

[33] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. E. Mohr. Chainsaw: Eliminating trees from overlay multicast. In *Proc. of International Workshop on Peer-to-Peer Systems (IPTPS'05)*, pages 127–140, New York, NY, February 2005.

[34] V. Vishnumurth and P. Francis. On heterogeneous overlay construction and random node selection in unstructured P2P networks. In *Proc. of INFOCOM'06*, pages 1–12, Barcelora, Spain, April 2006.

[35] X. Zhang, Q. Zhang, Z. Zhang, G. Song, and W. Zhu. A construction of locality-aware overlay network: moverlay and its performance. *IEEE Journal on Selected Areas in Communications*, 22:18–28, January 2004.

[36] B. Cohen. Incentives Build Robustness in BitTorrent. In *Workshop on Economics of Peer-to-Peer Systems*, pages 251–260, Berkeley, CA, May 2003.

[37] Napster Home Page. `http://www.napster.com/`.

[38] BOINC Home Page. `http://boinc.berkeley.edu/`.

[39] Freenet Home Page. `http://www.freenet.sourceforge.com/`.

[40] R. Hasan, Z. Anwar, W. Yurcik, L. Brumbaugh, and R. Campbell. A survey of peer-to-peer storage techniques for distributed file systems. In *Proc. of International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume II*, pages 205–213, Las Vegas, NV, April 2005.

[41] F. Pianese, J. Keller, and E. Biersack. Pulse, a flexible P2P live streaming system. In *Proc. of INFOCOM'06*, pages 1–6, Barcelona, Spain, April 2006.

[42] UUSee Home Page. `http://www.uusee.com/`.

[43] T. Locher, R. Meier, S. Schmid, and R. Wattenhofer. Push-to-pull peer-to-peer live streaming. In *International Symposium on Distributed Computing (DISC'07)*, pages 388–402, September 2007.

[44] Z. Li, Y. Yu, X. Hei, and D. Tsang. Towards low-redundancy push-pull P2P live streaming. In *Proc. of International ICST Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness (QShine'08)*, pages 1–7, Hong Kong, China, July 2008.

[45] R. Bindal, P. Cao, W. Chan, J. Medved, G. Suwala, T. Bates, and A. Zhang. Improving traffic locality in BitTorrent via biased neighbor selection. In *Proc. of IEEE International Conference on Distributed Computing Systems (ICDCS'06)*, pages 66–74, Lisboa, Portugal, July 2006.

[46] D. Choffnes and F. Bustamante. Taming the Torrent: A practical approach to reducing cross-ISP traffic in peer-to-peer systems. In *Proc. of ACM SIGCOMM'08*, pages 363–374, Seattle, WA, August 2008.

[47] C. Hsu and M. Hefeeda. ISP-friendly peer matching without ISP collaboration. In *International Workshop on Real Overlays and Distributed Systems (ROADS'08)*, Madrid, Spain, December 2008.

[48] X. Hei, C. Liang, J. Liang, Y. Liu, and K. Ross. A measurement study of a large-scale P2P IPTV system. *IEEE Transactions on Multimedia*, 9(8):405–414, December 2007.

[49] G. Kowalski and M. Hefeeda. Empirical analysis of multi-sender segment transmission algorithms in peer-to-peer streaming. In *Proc. of IEEE International Symposium on Multimedia (ISM'09)*, pages 243–250, San Diego, CA, December 2009.

[50] Y. Cai, A. Natarajan, and J. Wong. On scheduling of peer-to-peer video services. *IEEE Journal on Selected Area in Communications*, 25(1):140–145, January 2007.

[51] H. Shojania, B. Li, and X. Wang. Nuclei: GPU-accelerated many-core network coding. In *Proc. of IEEE INFOCOM'09*, pages 459–467, Rio de Janeiro, Brazil, April 2009.

[52] J. Li and C. Yeo. Content and overlay-aware scheduling for peer-to-peer streaming in fluctuating networks. *Journal of Network and Computer Applications*, 32(4):901–912, July 2009.

[53] SopCast Home Page. `http://www.sopcast.com/`.

[54] TVAnts Home Page. `http://www.tvants.com/`.

[55] P. Brucker. *Scheduling Algorithms*. Springer, 4th edition, 2004.

[56] C. Hsu and M. Hefeeda. Quality-aware segment transmission scheduling in peer-to-peer streaming systems. In *Proc. of ACM Multimedia Systems (MMSys'10)*, pages 169–180, Phoenix, AZ, February 2010.

[57] M. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer, 3rd edition, 2008.

[58] R. M'Hallah and R. Bulfin. Minimizing the weighted number of tardy jobs on parallel processors. *European Journal of Operational Research*, 160(2):471–484, January 2005.

[59] N. Magharei and R. Rejaie. Understanding mesh-based peer-to-peer streaming. In *Proc. of ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'06)*, pages 1–6, Newport, Rhode Island, May 2006.

[60] Video Traces Research Group, 2009. `http://trace.eas.asu.edu/h264/index.html`.

[61] Z. Liu, Y. Shen, K. Ross, J. Panwar, and Y. Wang. Substream trading: Towards an open P2P live streaming system. In *Proc. of IEEE International Conference on Network Protocols (ICNP'08)*, pages 94–103, Orlando, FL, October 2008.

[62] K. Graffi, S. Kaune, K. Pussep, A. Kovacevic, and R. Steinmetz. Load balancing for multimedia streaming in heterogeneous peer-to-peer systems. In *Proc. of ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'08)*, pages 99–104, Braunschweig, Germany, May 2008.

[63] Vuze Home Page. `http://www.vuze.com/`.