EMPIRICAL ANALYSIS OF MULTI-SENDER SEGMENT TRANSMISSION ALGORITHMS IN PEER-TO-PEER STREAMING

by

Greg Kowalski

B.Sc., University of Western Ontario, 2003

B.Eng.Sci., University of Western Ontario, 2003

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

Fall 2009

APPROVAL

Name: Greg Kowalski

Degree: Master of Science

Title of Thesis: Empirical Analysis of Multi-Sender Segment Transmission Algorithms in Peer-to-Peer Streaming

Examining Committee:

Dr. Tamara Smyth, Assistant Professor
Computing Science, Simon Fraser University
Chair

Dr. Mohamed Hefeeda, Assistant Professor
Computing Science, Simon Fraser University
Senior Supervisor

Dr. Qianping Gu, Professor
Computing Science, Simon Fraser University
Supervisor

Dr. Joseph Peters, Associate Director, Professor
Computing Science, Simon Fraser University
Examiner

Date Approved: December 10, 2009

Declaration of
Partial Copyright Licence

Abstract

We study and analyze segment transmission scheduling algorithms in swarm-based peer-to-peer streaming systems. These scheduling algorithms are responsible for coordinating the streaming of video data from multiple senders to a receiver in each streaming session and they have not been rigorously analyzed in the literature. We first conduct an extensive experimental study to evaluate various scheduling algorithms on many PlanetLab nodes distributed all over the world. We study three important performance metrics: continuity index which captures the smoothness of the video playback, load balancing index which indicates how the load is spread across sending peers, and buffering delay required to ensure continuous playback. Then, we propose a new scheduling algorithm called On-time Delivery of VBR (Variable Bit Rate) streams. Our experiments show that the proposed scheduling algorithm improves the playback quality by increasing the continuity index, requires smaller buffering delays, and achieves more balanced load distribution across peers.

Keywords: peer-to-peer streaming; segment scheduling; multi-sender transmission

Acknowledgments

This thesis has taken me over 4 years to complete, mainly due to the fact that I have been working full-time throughout. My recommendation to anyone is to avoid doing anything similar because it has the potential to impose great stress on the personal life. I was lucky to have excellent support of my family, including my dad Pawel, my mom Halina, and my sister Magda. Also, my company was very supportive of my education by covering all of my tuition fees over the 4 years, thanks to Manuel and Chris. Finally, I received great help and motivation at the right times from my senior supervisor, Mohamed Hefeeda. Thank you all.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Peer-to-peer (P2P) file sharing systems have gained tremendous popularity in the past few years. More users are continually joining such systems and more objects are being made available, motivating other users to join. The traffic generated by P2P systems accounts for a major fraction of the Internet traffic today [26], and is bound to increase [34].

P2P systems were introduced keeping in mind the limitations of client-server architecture in an Internet-scale distributed environment (such as World Wide Web). In P2P systems, every node acts both as a client and a server and contributes by providing access to its computing resources. It follows three main principles which provide a generic overview of any P2P framework.

1. The principle of resource sharing: all P2P systems involve an aspect of resource sharing, where resources can be physical resources, such as disk space or network bandwidth, as well as, logical resources, such as services or different forms of knowledge. Resource sharing gives applications more power and capability than can be provided by a single node.

2. The principle of decentralization: this is an immediate consequence of resource sharing. Parts of the system or even the whole system are no longer operated centrally. Decentralization is particularly interesting in solving the problems of single-point-of failures or performance bottlenecks in the system.

3. The principle of self-organization: when a P2P system becomes fully decentralized,

there no longer exists a node that can centrally coordinate all of the system's activities or a database that can store all global information about the system centrally. Therefore nodes have to self-organize themselves, based on whatever local information is available and interact with locally reachable nodes (neighbors). The global behavior then emerges as the result of all the local behaviors that occur.

With the increasing number of P2P file sharing systems being deployed on the Internet, P2P streaming technology has become very popular over the past several years. Many P2P streaming systems have been proposed in the literature and deployed in real life [40, 41, 52, 61, 66]. Among these systems, mesh-based (also known as swarm-based) systems are simpler to implement [2] and more widely used in practice; examples of such systems include CoolStreaming [71] and PPLive [50]. Mesh-based systems also adapt better to network dynamics, lead to better perceived quality [44], and incur lower maintenance overhead. The goal of this research is to further improve the performance of mesh-based streaming systems. Such systems typically have several main components for overlay management, allocation of seed server resources, peer selection for forming swarms, and coordination of senders in each streaming session. Most of these mesh-based systems employ pull-based streaming protocols.

In particular, in mesh-based systems, a video stream is partitioned into small segments, and segments are transmitted from multiple senders to a receiver. The receiver uses a segment scheduling algorithm to compute the transmission schedule for each sender. The schedule specifies which segments to send and their transmission times. The scheduling algorithm is an important component that directly impacts the user-perceived visual quality in the streaming session [28]. Despite the importance of segment scheduling algorithms, they have not been rigorously analyzed in the literature. For example, works such as [6, 12] only use simulations to evaluate the scheduling algorithms. Simulations, although useful for preliminary analysis, may abstract away many important practical details.

It is important to understand the importance of segmentation and the need for segment scheduling. Owing to their large sizes, video objects have to be segmented. Segmentation also enables a client to receive different portions of the same object from different senders at the same time. Unlike file sharing, P2P streaming imposes timing constraints on the data transfer from the senders to the receiver. This means that every segment should have a deadline for arrival at the receiver as too many segments missing their deadlines will

decrease the user-perceived quality. This necessitates the need for scheduling that will assign segments to peers to decrease the completion time and efficiently utilize system resources such as network bandwidth.

1.1 Problem Statement and Contributions

We are faced with a problem of delivering high quality video from multiple senders to a receiver. In streaming scenarios, the entire video is not always available at every peer and it is often inefficient to transmit the entire video from a single source peer (for example, that could lead to overloading of the source peer). The elegance of streaming is obviously the fact that we do not need to have the entire video downloaded before the playout begins. We can split the video file into well defined segments, identify other peers that have the required segments available, request these segments from those peers, receive them (including some buffering) and play them out. This thesis deals mainly with the details concerning the requesting of video segments from other peers.

It is important that we provide a continuous supply of the video segments once the playout begins. If only a few segments are missing once in a while, the video quality will still be acceptable by most standards. If too many segments are missing the video and the audio quality will suffer and the media will not be useful. A missed segment does not always indicate network failure (due to congestion or other problems), or unavailability of this segment within the network. It may simply mean that the segment was not delivered on time to the receiver, i.e., the segment deadline was missed. The main problem that we are solving is creating a schedule that minimizes the number of video segments missing their deadlines.

In a typical scenario, there will be several sender peers and one receiver peer. The network characteristics of each peer, including the bandwidth of each sender peer will be known because the system will continually estimate them. Also, the segment availability information will be continually shared among the neighbouring peers. We assume that each peer holds one or more sets of contiguous ranges of segments. We make this assumption because the most common scenario for the users is to watch the entire video from the beginning. A less typical case occurs when users scan through the video searching for a particular piece. The above user behaviors result in the receivers requesting one or more portions of the video of varying length. It should also be noted that most implementations,

including ours, TCP is used as the underlying transport layer for the data transmission. This implies that all segments will be delivered to the receivers but some of them may arrive late. All peers, however, store all received segments including the late segments so that they can be used for sharing in later streaming sessions. This ensures that all peers store contiguous ranges of segments.

We will also know the network characteristics of the receiving peer and most importantly the maximum incoming bandwidth of the receiver. Finally, we will have the details of the media file that the receiver is requesting including the number and the sizes of segments. We cannot schedule all the sender peers to send at their maximum bandwidth to the receiver in case their total bandwidth is greater than the incoming receiver bandwidth and this would flood the receiver's network resulting in many discarded video segments.

In summary, we will have the following information:

- A set of senders;
- A set of available segment ranges at each sender;
- Bandwidth estimate of each sender;
- Maximum incoming bandwidth of the receiver;
- A set of segments that are requested by the receiver;
- Lengths of all segments requested by the receiver;
- Deadlines of all segments requested by the receiver.

The problem addressed in this thesis is called the Multi-Sender Segment Transmission Schedule (MSTS) problem, and it can be stated as follows. We are given a set of n video segments, $S = \{s_1, s_2, \ldots, s_{\mathrm{n}}\}$, with an associated set of segment lengths, $L = \{l_1, l_2, \ldots, l_{\mathrm{n}}\}$, and a set of segment deadlines, $D = \{d_1, d_2, \ldots, d_{\mathrm{n}}\}$. We also have a set of m peers, $P = \{p_1, p_2, \ldots, p_{\mathrm{m}}\}$ with an associated set of peer bandwidths, $B = \{b_1, b_2, \ldots, b_{\mathrm{m}}\}$. Finally, we are given the set of segment ranges available at each peer. Since typically peers hold contiguous ranges of segments, we assume that the segment range at each peer is given by the highest segment number available. We assume that all segments with lower segment numbers are also available at that peer. The set of available segment ranges is given by $A = A_1 \cup A_2 \cup \cdots \cup A_{\mathrm{m}}$. We need to create a schedule J where each segment has an

assigned peer and transmission start time such that the segment will be transmitted to the receiver before its deadline. Since this may not always be possible and missed segments must be taken into consideration, we want to minimize the number of segments that miss their deadlines. We have the additional constraint that the schedule cannot exceed the incoming receiver bandwidth $B_{\max}$.

We assume that at the time of scheduling all of the above parameters are constant. These parameters may change over the course of a video streaming session but we handle this by creating a new schedule using a more recent snapshot of the system and network state. We must make sure, however, that this can be done efficiently. The first contribution of this thesis is to show the hardness of the segment scheduling problem by demonstrating a polynomial reduction from the parallel machine job scheduling problem [22].

Theorem 1.1.1 *Multi-Sender Segment Transmission Schedule problem is NP-Hard.*

Proof: We will show that a restricted version of the *Multi-Sender Segment Transmission Schedule* (MSTS) problem is as hard as *Multiprocessor Scheduling* (MS), an NP-complete problem [22]. Recall, that in an instance of the MS problem, we have a set T of tasks, number $m \in Z^+$ of processors, length $l(t) \in Z^+$ for each $t \in T$, and a deadline $D \in Z^+$. Is there an m-processor schedule for T that meets the overall deadline D, i.e., a function $\sigma : T \rightarrow Z_0^+$ such that, for all $u \geq 0$, the number of tasks $t \in T$ for which $\sigma(t) \leq u < \sigma(t) + l(t)$ is no more than m and such that, for all $t \in T, \sigma(t) + l(t) \leq D$? [22]

We reduce the MS problem to a simpler instance of MSTS problem as follows. Tasks are segments. We set the deadlines for all segments as D. All sending peers have the same bandwidth, and each can only transmit one segment at a time. Segment lengths are set as tasks lengths. We assume all segments are available at each peer. Now, clearly, deciding whether a schedule exists for MSTS would solve the MS problem. □

If an algorithm takes a very long time to compute an optimal solution, it is not very useful for the purposes of video streaming where we may need to rerun the scheduling algorithm many times (due to changing network conditions and/or peer membership changes). Since, we cannot afford the time to compute the optimal solution, we settle for a practical solution with good performance.

The second contribution of this thesis is an extensive experimental study to evaluate various scheduling algorithms in the literature [36]. We isolate the scheduling algorithm from the whole P2P streaming system and highlight its impact on the overall system performance.

We implement the three most common scheduling algorithms in a streaming prototype and we deploy our prototype on more than 70 PlanetLab nodes distributed all over the world. We use actual video streams with diverse visual content, motion complexities, number of frames, and bit rates in the experiments. We study three important performance metrics:

1. Continuity index, which captures the smoothness or the user-perceived quality of the video playback;

2. Load balancing index, which indicates the degree of resource sharing across the sender peers; and

3. Buffering delay required to ensure continuous playback.

Our analysis provides insights on the functioning of scheduling algorithms, and highlights the strengths and weaknesses of each algorithm. This is useful for other researchers working on optimizing the performance of P2P streaming systems.

Based on our analysis, we propose a new scheduling algorithm, which we call On-time Delivery of Variable bit rate streams (ODV), which is the third contribution of this thesis. A key feature of the proposed algorithm is that it considers the variability in the bit rates of encoded video streams, which is more realistic as most encoded videos have VBR because of the different compression methods used for different frame types and the diverse visual complexities of video frames. That is, unlike previous scheduling algorithms that typically assume segments have fixed size computed using the average bit rate of the video stream, our algorithm allows scheduling of video segments with variable sizes. Our experimental results show that the proposed ODV scheduling algorithm improves the playback quality by increasing the continuity index, requires smaller buffering delays, and achieves more balanced load distribution across peers.

1.2 Thesis Organization

The rest of this thesis is organized as follows. In Chapter 2, we provide a brief background on P2P systems and summarize the related work. In Chapter 3, we describe and analyze the current scheduling algorithms. Our proposed P2P scheduling algorithm is described and qualitatively compared to other common scheduling algorithms in Chapter 4. We evaluate

and compare the performance of all scheduling algorithms in Chapter 5. Chapter 6 concludes and outlines future directions for this work.

Chapter 2

Background and Related Work

This chapter provides general background on peer-to-peer (P2P) technologies and previous work related to segment scheduling in P2P streaming systems. We start in Section 2.1 by discussing different P2P architectures and applications, leading into a more detailed discussion on P2P streaming approaches. We discuss the strengths and weaknesses of various P2P streaming architectures, and we describe where our work fits in. Section 2.2 describes previous works and how they are different from our research.

2.1 Background on P2P Systems

In the recent years, P2P systems have been receiving increasing attention [16,45] due to their ability to combine independent nodes into a shared pool of resources. A major breakthrough occurred in 1998 when file sharing P2P applications such as Napster [16] gained tremendous popularity. Napster took North America by a storm with millions of active users and Terabytes of shared data. According to [26, 55], the P2P paradigm continues to grow and its bandwidth on the Internet has been higher than WWW for several years. Let us start, however, by defining what a P2P system actually is.

A P2P system is a distributed network of independent nodes, called peers, where each node contributes a portion of its resources such as CPU power, network bandwidth, data or disk storage, to all other system nodes. A pure P2P system does not have a need for any central servers or coordination instances [56]. Figure 2.1 shows a generic P2P system where all peers can act as either consumers or suppliers of system resources, and other than the different amounts of resources they contribute, nothing else separates the peers

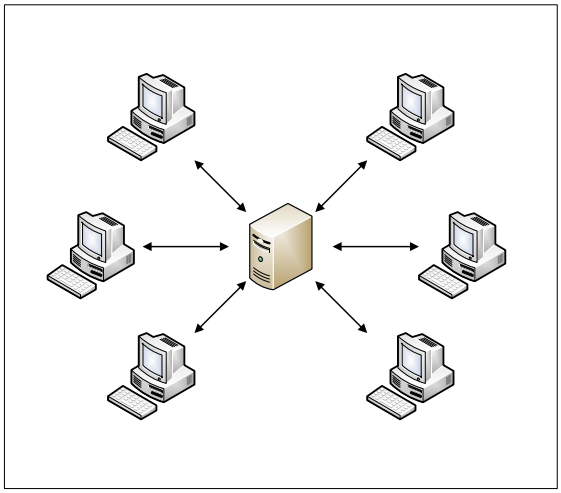


Figure 2.1: Centralized server-based service model.

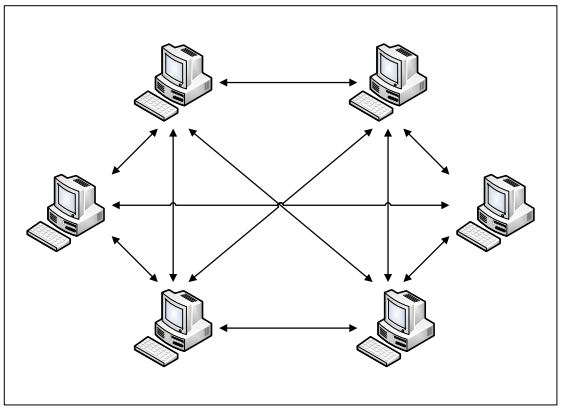


Figure 2.2: A P2P system of nodes without central infrastructure.

from each other. Figure 2.1, on the other hand, shows the traditional client-server model, where we have at least two types of nodes: dedicated servers who supply resources and clients who consume them. P2P systems provide many benefits over traditional client-server applications, including better scalability, robustness, and cost-effectiveness.

2.1.1 Architecture of P2P Systems

P2P systems are composed of dynamically created networks where nodes can join and leave at any time without having a major impact on the system. These dynamics of peer participation, called churn, are an inherent property of P2P systems. This distributed

architecture provides more scalability, since there are no theoretical upper bounds on the number of nodes in the system and stronger overall robustness, since there is no single point of failure.

A pure P2P system does not have clients or servers; there are only peers, which function as clients and/or servers to other peers depending on the instantaneous system state. This is different from traditional client-server architectures where communication always flows to and from a central server. An example of such a traditional client-server application is the FTP file transfer server where the clients always initiate the upload or download operation while the server reacts to satisfy these requests.

Although various approaches were attempted, in most cases, the P2P system forms an application layer overlay network on top of an existing physical topology, using the existing network and transport level protocols such as IP and TCP. This P2P overlay network is composed of all currently participating peers and it is used to provide various P2P system services such as indexing and peer discovery.

P2P systems can generally have one of two overlay network types: structured and unstructured. A structured P2P overlay normally has static connections and usually uses distributed hash table (DHT) based indexing for data lookup. Conversely, an unstructured P2P overlay has no controlled organization and no optimizations on connections. These concepts are described in more detail in the following sections.

2.1.2 Structured P2P Systems

One of the main goals for a structured P2P system is to provide a global protocol that guarantees successful resource route searches. This functionality requires more structure in the overlay links that is most commonly achieved using DHTs, which assign ownership of resources to peers. This is similar to a regular hash table assigning keys to array slots. The following sub-section describes the use of DHTs in P2P systems in more detail.

Distributed Hash Tables

Distributed hash tables are a class of decentralized distributed systems that provide lookup services for key-value pairs. As shown in Figure 2.3, in the case of P2P systems, the keys may be data file names and the values may be peer locations. The important detail we should note is that this key-value mapping responsibility is distributed among all peers and thus

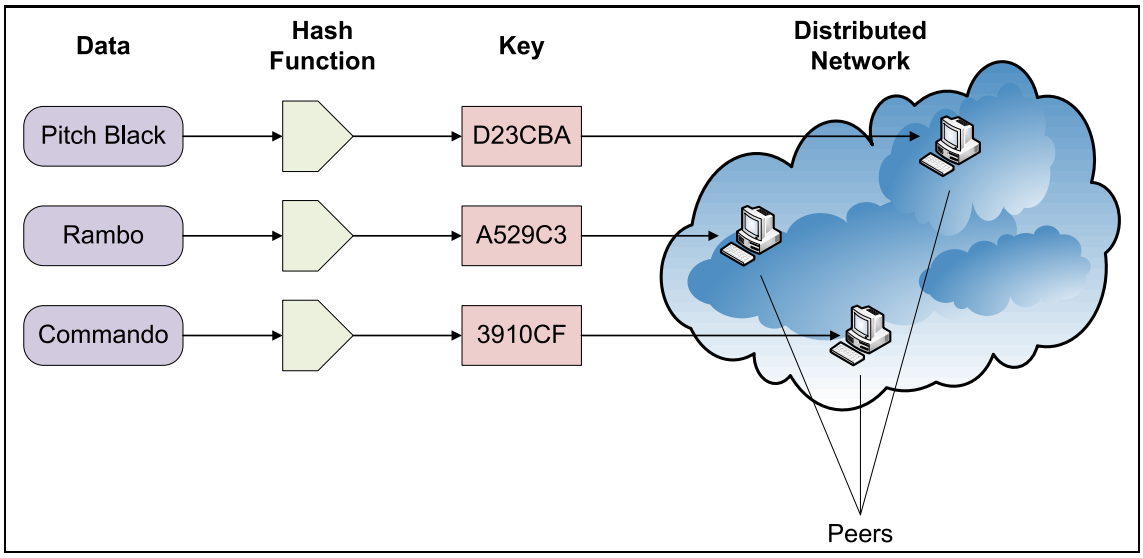


Figure 2.3: Distributed Hash Table.

churn causes very minimal disruption in this lookup service. Some of the existing networks that use DHTs are BitTorrent (distributed tracker) [5], Kad network [32], YaCy [67], and Coral Content Distribution Network [14].

DHTs are also used in applications that require efficient resource discovery, such as grid computing, for the purposes of resource management and scheduling of applications. Specifically, DHTs can be used to find resources that match user application requirements [29,68]. Also, some more recent advances in decentralized resource discovery extend existing DHTs to support query routing and multi-dimensional data organization [3].

2.1.3 Unstructured P2P Systems

In an unstructured P2P system, the overlay network links are established in an ad-hoc manner. There is no need to store the structure data because it is easy to create new links. New peers can copy links to others from existing peers as well as form their own links over time. Searching for resources in an unstructured P2P system, however, requires flood queries to be propagated through the network to find as many peers as possible who may have the required resource. The main disadvantage of this approach is that queries cannot always be resolved since it is not practical to reach the entire network. Therefore, some of the resources available in the network may not always be found. In terms of practical

implications, however, popular content is usually available throughout the overlay since more peers are likely to search for the popular resource but rare data may be more difficult to find. Another downside of the flood searching approach is the increased network traffic and consequently poor search efficiency.

There are three general models of unstructured P2P systems: pure, hybrid, and centralized. A pure architecture is composed only of equivalent peers, without a central server managing the network and without a central router. Gnutella [25] and Freenet [21] are examples of such decentralized systems. As the name implies, the hybrid architecture, is a combination of equivalent peers and some special peers called supernodes. The role of the supernodes is to provide some overlay network management services such as resource lookup. Regular peers can also become supernodes depending on the momentary needs from the network. Centralized P2P architectures require a central server for bootstrapping the system itself and also providing resource indexing functions. This is similar to structured P2P systems, however, peer connections are not managed by the central server and they are still formed in an ad-hoc manner. The main disadvantage of centralized systems is that they will not work without the central servers, which also are points of failure. The main advantage of the centralized architecture is the improved efficiency and effectiveness of queries because resources are indexed by the central server. Examples of unstructured, centralized P2P systems include Napster [46] and eMule [19].

2.1.4 P2P Applications

A standalone P2P system by itself does not provide any specific practical function. In most cases, applications are build on top of P2P systems where the P2P system provides the application with services such as peer management, and resource searching and indexing. The following list, although not exhaustive, describes various types of applications built on top of P2P systems.

File Sharing

The simplest and most popular application of P2P systems is file sharing. The system provides indexing services to allow the discovery of peers who have the files requested by others. This type of application only stores and serves files to peers and most common file sharing applications use unstructured P2P architectures. The reason for this choice of

architecture is that unstructured systems are more efficient at keyword searching and are more resilient to high peer churn. Examples of such applications existing today are Kazaa [35], BitTorrent [5], Gnutella [25], uTorrent [62], iMesh [30], eMule [19], and Limewire [39].

File and Storage Systems

In this type of application, a distributed file system is constructed from an overlay network of peers. The system provides functions such as access control, directory structure, data integrity and consistency, caching, and replication. These requirements usually require structured P2P architectures so that guarantees and efficiency of file lookup can be provided. The advantages of such distributed P2P file systems include cost-effectiveness since the required resources are already deployed, high availability and huge storage capacity since the number of peers in the network can be high. Examples of such applications are cooperative file system (CFS) [18] on top of Chord [17], OceanStore [20, 51] on top of Tapestry [72] and Past [54] on top of Pastry [53].

Distributed Cycle Sharing

Another, perhaps less well-known application of P2P systems is the distributed cycle sharing. In these types of applications, the resource provided by the network is the CPU processing power where peers can complete smaller parts of a larger processing process. Not all types of problems can benefit from this approach since each problem needs to be easily decomposed into smaller, largely independent sub-problems. Some interesting examples of these types of applications include SETI@Home (Search for Extraterrestrial Intelligence) [57] where the processing problem involves analyzing signals received by radio telescopes to determine whether an intelligent life exists outside of earth. Another example is the Genome@Home [23] project, whose goal is to study and understand human genetic information.

In each of these systems, there is a central manager that distributes the work and collects the results from all peers. These systems are not fair because although the peers contribute CPU power when idle, they do not get direct benefits by submitting their own computational problems. Some systems for sharing CPU cycles among peers have been proposed, such as CPUShare [15], and they usually use structured P2P architectures to locate peers with free CPU cycles that can be shared by others.

Media Streaming

Media streaming is a more recent application of P2P systems. While regular file sharing systems can provide media content, they achieve this by first downloading an entire file and then playing it back. Streaming systems require only a short buffering delay, in the order of seconds, before the media playback can begin. There are many existing streaming applications but since they are the main focus of this thesis, they will be evaluated in more detail in Section 2.2.

2.1.5 P2P Streaming

There are two main categories of P2P streaming systems: live and on-demand. A live P2P streaming system is analogous to a TV broadcast and implies that the streamed content must be played back at the clients as close to real-time as possible. In this type of application we normally have a single media stream and all clients are synchronized. An on-demand streaming system, on the other hand, mostly resembles the playback of recorded media such as VCR or DVD. Depending on specific system requirements, navigation functions such as pause, rewind and forward may need to be provided. In this type of system, the clients may individually choose the timing of the media playback and thus their streams are most likely not synchronized. In addition to live and on-demand P2P streaming system categories, we can also distinguish two overlay network types: tree-based and mesh-based.

Tree-based Systems

In tree-based streaming systems, we usually have a single source peer that propagates the data stream to its children who then propagate it to their children and so forth. In the past, IP multicast was proposed as a solution to this problem, however, due to the prohibitively large router overhead for managing multicast groups and the complexity of transport control for multicast sessions, this solution was never widely deployed. Instead, two approaches that utilize an application level overlay network are used: single-tree and multi-tree streaming.

Single-tree streaming systems form trees at the application level similar to IP multicast trees formed by routers at the network level. As shown in Figure 2.4, the root node of the tree is the media content source server and the peers can join the tree at various levels. The tree is structured and the peers receive data from their parent as well as send the data to their children. Every peer is either an interior node that receives and sends data or a

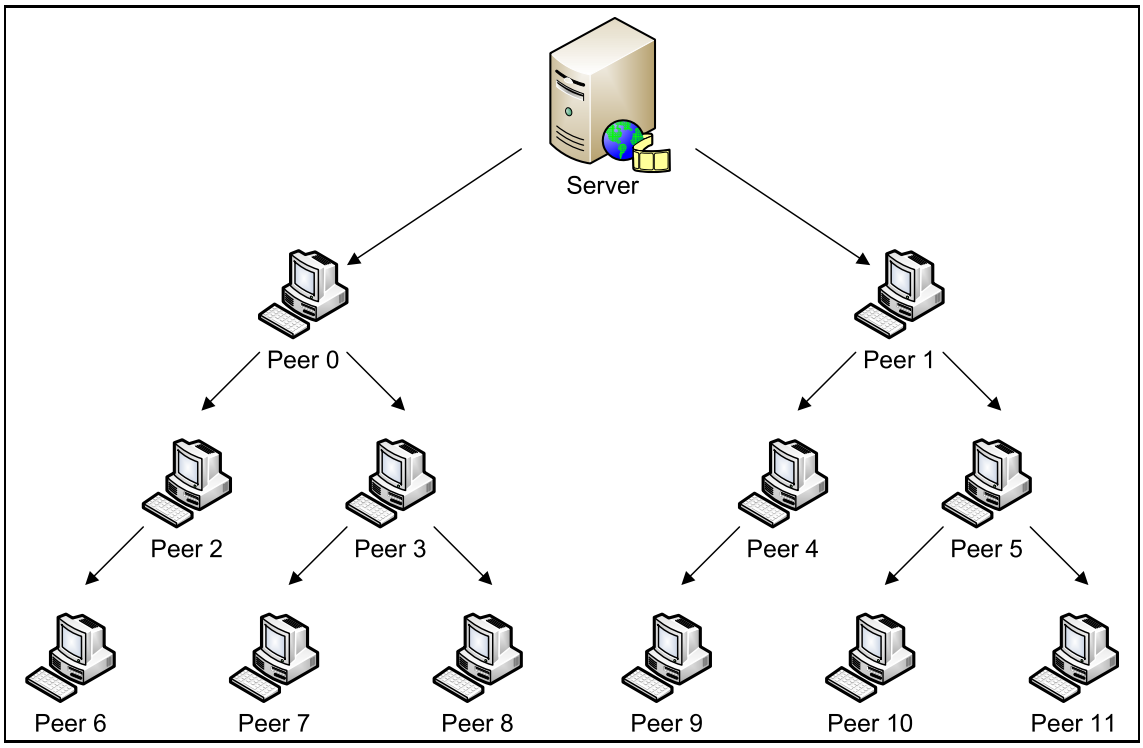


Figure 2.4: Application layer multicast tree for P2P video streaming.

leaf node that only receives data. This architecture has two major problems: (i) it is not fair since the number of leaf nodes increases faster than the number of interior nodes, and (ii) interior nodes may not be able to handle high-bandwidth applications such as high-quality video due to network capacity limitations. This approach can cause scenarios where the children of such interior nodes are limited by their parent's insufficient bandwidth. In most cases, the outgoing bandwidth of these children cannot be fully utilized, which can lead to inefficient use of resources. Examples of single-tree based streaming systems include Overcast [31] and End System Multicast (ESM) [13].

When dealing with single-tree streaming systems, we should consider two aspects: depth of the tree and fanout at each level. The greater the depth of the tree, the larger the number of peers for the stream to travel causing larger delays. The larger fanout, although advantageous, is limited by the outgoing peer bandwidth. Also, it should be noted that tree-based streaming systems have poor response to high peer churn because every time a peer leaves the network, all of its children get disconnected from the stream. Although there

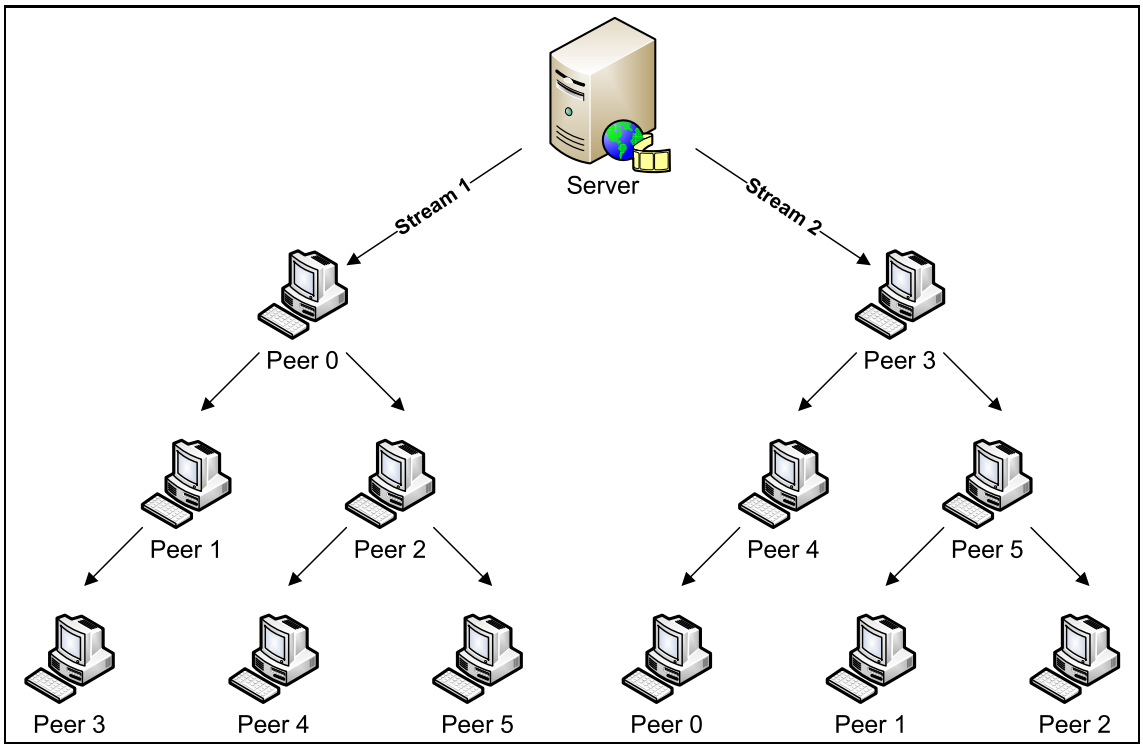


Figure 2.5: Multi-Tree based streaming with two sub-streams and six peers.

are many algorithms for this problem [60], tree-based streaming systems still cannot recover fast enough to handle frequent peer churn.

Multi-tree streaming systems were designed to overcome the problems with single-tree systems [7, 33]. In these types of systems, the main media stream is divided into several sub-streams and multiple trees are formed, one for each sub-stream. Within each sub-tree, the behavior is similar to single-tree systems. Each peer, however, has a position at different levels within different sub-trees, i.e., a peer can be an interior node in one sub-stream tree and it can be a leaf node in another sub-stream tree. Figure 2.5 shows an example of a multi-tree streaming system. Multi-tree systems provide improved robustness over single-tree systems since node failure causes only a single sub-stream to be disconnected. Also, since peers have different positions within various sub-stream trees, most peers will contribute bandwidth to the streaming session.

One of the disadvantages of this approach is that there is no guarantee of finding an

optimal forest, even with sufficient capacity because it is an NP-Complete problem. Tree-based systems were the first attempt at the media streaming problem and although many successful applications have been deployed, they still have problems and thus alternative solutions were proposed. A primary example of such alternative solution to the tree-based media streaming problems are the mesh-based overlay networks discussed in the following section.

Mesh-based Systems

In tree-based systems, every peer has one parent for the video stream (or sub-stream in case of multi-tree) and, as mentioned before, this creates a single point of failure. If the parent leaves the network, all of its children are disconnected from the stream making it is extremely difficult to manage a tree with high churn. Mesh-based systems with no static topology and a dynamic structure are an alternative. In these types of systems, any peer can upload to and download from multiple peers simultaneously. This high peering degree makes the mesh very resilient against peer churn. Studies have been performed that prove the performance of mesh-based systems is better than the tree-based systems [44].

Some of the central functionality within mesh-based system is the mesh formation and maintenance. In most cases, trackers are used for getting an initial list of peers and for contacting some of the active peers. Once links to a sufficient number of peers are formed, the data exchange can start. In order to deal with churn, peer lists get refreshed periodically or alternatively trackers can be contacted to get an updated list of active peers. When a peer leaves the overlay network gracefully, it notifies its neighbors so that it can be replaced immediately. In the case of a peer crash, the keep-alive messages that peers send to their neighbors will time out, notifying the peer that its neighbor is no longer available.

In order to start a streaming session, a peer must select a group of its neighbors to download data from. This group is usually selected based on criteria similar to the following: (i) resource availability at both ends, i.e., the number of connections, bandwidth, CPU and memory usage, (ii) connection quality, delay and data loss rates, and (iii) content availability. Once a group of peers has been selected based on the above criteria, data exchange can commence. During the data exchange, the media is divided into segments and the receiving peer will request various segments from different peers. The segments are likely to arrive at the receiver out of order and therefore they must be buffered before they can be played back.

There are two main methods for distributing media content: push and pull. In push-based distribution, peers push media content to their neighbors but since no parent-child relationships are defined, the push method may propagate redundant segments. In pull-based distribution, peers exchange buffer maps, i.e., segment availability, and use this information to create efficient schedules for data exchange. This thesis focuses on this pull-based data exchange and specifically examining the effects of different segment scheduling algorithms on the resulting stream quality. The following section examines some previous research on the different approaches to the segment scheduling problem.

2.2 Related Work

A recent measurement study on PPLive [50] reports that users suffer from long start-up delays and playout lags, and suggests that better segment scheduling algorithms are needed [28]. The main goals of scheduling algorithms in P2P streaming systems are to achieve a target quality of service and balance the load on peers. Constructing *optimal* segment schedules to maximize the video quality, however, is computationally complex (NP-Complete) and therefore, many P2P streaming systems, such as [1,47,71], resort to heuristic algorithms for segment scheduling. The authors of [47] propose to randomly schedule segment transmission. The authors of [71] assume that segments with fewer potential senders are more likely to miss their deadlines, and propose to schedule the segments with fewer potential senders earlier. The authors of [1] describe a weighted round-robin algorithm based on senders' bandwidth. Unlike our algorithm, none of these algorithms considers the segment scheduling problem with VBR video streams.

The authors of [8] formulate an optimization problem to maximize the perceived quality, and they solve it using an iterative descent algorithm. The authors of [70] 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 this segment. They then transform the segment scheduling problem into a min-cost flow problem. These algorithms, however, are computationally expensive and can not be used in *real-time* streaming systems in which peers typically have limited resources to solve these optimization problems. Therefore, we do not consider these algorithms further in this thesis.

Several other works are also related to the segment scheduling problem, but they do

not directly solve it. The authors of [6] 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. The authors of [2] propose using network coding to bypass the scheduling problem among small blocks belonging to the same relatively large segment. However, employing network coding may impose higher processing overhead on peers, which may require special hardware to speed the decoding process [59], and is not easy to deploy. Finally, several segment scheduling algorithms, such as [38], have been proposed for tree-based systems. They are, however, not applicable to mesh-based systems, in which peers have no knowledge on the global network topology.

The ultimate requirements of a P2P media streaming system are a continuous, high quality playback, a low buffering delay, and a balanced utilization of system resources. Streaming parameters such as the number of senders and the streaming bit rate can have a dramatic effect on the system performance and it is essential to evaluate their effects. As we will demonstrate in Chapter 5, these parameters can have significant effects on the scheduling algorithm performance. In some cases, the choice of the scheduling algorithm can affect the playback quality by 25% and change the buffering delay 50 times.

Very few existing systems compare their scheduling algorithms to the existing ones or even provide clear justification for their scheduling algorithm selection criteria. Even real-life systems such as CoolStreaming [71] and PPLive [50] fail to compare their scheduling algorithms with others. Although few papers do compare the scheduling algorithms, their experiments are limited to computer simulations [70] and no Internet experiments have been performed. To the best of our knowledge, no thorough empirical analysis of various scheduling algorithms has been performed and validated using either real life deployments or the PlanetLab environment.

It should also be noted that there currently exists no framework for comparing scheduling algorithms. The development of a P2P scheduling algorithm comparison system can be fairly complex and different design decisions can all have effects on the resulting scheduling algorithm performance. This implies that comparing scheduling algorithm by using independently developed systems can be quite inaccurate so it is advantageous to develop a framework that strictly concentrates on comparing the relative performance of different scheduling algorithms. Such frameworks can aid the evaluation of the effects of the streaming parameters such as the number of senders and the streaming bit rate. Our system, as described in Chapter 5, is an implementation of such a framework. It greatly improves

the speed and quality of our research by providing an efficient deployment, testing and evaluation process for new scheduling algorithms. We shall now describe the advantages and disadvantages of various segment scheduling approaches, starting with one of the most popular P2P streaming systems, CoolStreaming [71].

The DONet [71] protocol, used by CoolStreaming system, is one of the strongest protocols that has been tested and deployed in the real world. It is a data-driven overlay network and its fundamental aspects are that every node periodically exchanges data availability information with a set of partners, and retrieves unavailable data from one or more partners, or supplies available data to partners. It is easy to implement, since it does not require having to construct and maintain a complex global structure. It has proved to be efficient, as data forwarding is dynamically determined according to data availability. It is robust and resilient, as the partnerships enable adaptive and quick switching among multiple suppliers.

DONet proposes a member and partnership management algorithm, and a rarest first scheduling algorithm. The control overhead and transmission delays are kept low. The performance of the simulation and the algorithm has been evaluated extensively on PlanetLab. The real implementation called CoolStreaming was released May 30, 2004 and had over 30000 distinct users. The implemented algorithm was deployed in real life and usage statistics were provided hence the popularity of the product. The streaming quality results were given based on both the PlanetLab tests and real-life deployments. This protocol employs a simple rarest-first scheduling algorithm that is fairly easy to implement.

A few negative aspects of the DONet protocol were the failure to mention the reasoning for choosing rarest-first scheduling algorithm. This work failed to mention if any other scheduling algorithms were considered while performing the simulations. There was also no account of how bandwidth and deadlines were estimated. There was only an indication that videos streamed were CBR and there were no comparisons with other algorithms to indicate the performance evaluation of the scheduling algorithm used. The results specify the continuity index for the overlay network but metrics that measure distribution of the load among the supplier peers was not included. Nevertheless, this is one of the most important works to date and serves as a solid basis for comparison against other algorithms.

As mentioned earlier, another interesting solution models the scheduling problem as a min-cost network flow problem [70] where two potential solutions are proposed: a global optimal algorithm and an approximate distributed algorithm. The global algorithm is not practical since it requires global knowledge of the entire overlay network. The distributed

algorithm is the global algorithm's practical version that makes some approximations based on local assumptions. These two algorithms were compared to DONet, Round-Robin, and Chainsaw (random) algorithms using computer simulations.

When determining the block priority, the proposed optimal algorithm combined both rarest-first and earliest deadline factors. Some of the positive aspects of this approach lie in the fact that it employs a theoretically optimal algorithm and it also provides an efficient heuristic algorithm that the experiments have proven superior to others.

The negative aspects of this work include the usage of a complex algorithm that is not easy to implement. Also, only a computer simulation was performed and the algorithms were not tested in a real-life system like PlanetLab. There was no mention of the real goal of the scheduling, i.e., to minimize the number of missed segments and the real user behavior was not taken into account. This solution failed to show the effects of different number of senders and the maximum streaming bit rate in the simulations was only 500 Kbps.

According to [71], the DONet algorithm has a continuity index of approximately 0.98 when streaming 500 Kbps with 4 or more senders but this work claimed that the DONet algorithm only reaches approximately 0.80 continuity index when streaming at 500 Kbps, which shows an inadequacy of this computer simulation. There was no in-depth analysis of the streaming sessions and important performance metrics such as fairness index or sharing index were not mentioned.

Another interesting work, focuses on transmission scheduling of the media data for a multi-source streaming session and it realizes a sophisticated scheduling scheme, called fixed-length slotted scheduling (FSS), which results in minimum buffering delay [37]. FSS employs variable length segments whose size is determined based on the bandwidth at which each segment is transmitted.

The negative aspects of this work are that it is claimed that the P2P system is self-growing since the requesting peers can become supplying peers only after they receive all the data [37] but this is not true because a peer does not require all the data to become a supplier. This approach uses buffering delay as the only performance metric, however, buffering delay is not as important as other metrics such as quality and continuity index, which were not mentioned.

Another problem is that keeping the buffering delay at a minimum does not always result in the best streaming experience. It was indicated that FSS schedules the video segments really close to their deadlines to minimize the buffering delay. This is an impractical

evaluation and if this scheduling scheme was applied in practice, it would likely not deliver quality streams. It has been observed from PlanetLab experiments that due to network fluctuations, the bandwidth estimates change frequently and this could have a detrimental effect on the performance of FSS. Finally, this work did not include any real-life experiments or even computer simulations.

The Network Coding (NC) technology, used to achieve maximum data flow in a network, has also been used as the basis for some scheduling algorithms. One example is the generic buffer-assisted search (BAS) scheme that improves partner search efficiency by reducing the size of index overlay [12]. This scheme is used to create a novel scheduling algorithm based on Deadline-Aware Network Coding (DNC) to fully exploit network resources by dynamically adjusting the coding window size. The extensive simulation results shown, demonstrate that BAS can provide a faster response time with reduced control cost as compared to the existing search methods. Also, DNC improves the network utilization and provides high streaming quality under different network conditions. This approach models the scheduling problem as a network flow problem and carries out its scheduling scheme with that assumption.

The positive aspects of this work include that fact that segment schedules are created based on more than just local decisions. This solution also claims that Round-Robin algorithm, smallest-delay model, and a rarest first, pull-based gossip (PGA) model use simple heuristics whose local decisions make inefficient use of network capacities. An optimal network coding algorithm is presented along with a distributed protocol. Strong theoretical proofs of the algorithms have been depicted and it has been proven that DNC and NC algorithms outperform PGA with respect to network throughput (average number of segments a peer receives per second). The negative aspects of this work are that it fails to address to VBR videos and it assumes each segment to have a uniform one second long play-out time.

There are also a few practical concerns regarding the implementation of the NC algorithm for streaming. It was noticed that the system needed to use more peers in order to fully distribute the segments, i.e., the senders need to receive data from other senders and pass it to the receiver. In the simulation, all links had a bandwidth of 3 Mbps or more, while the streaming bit rate was only 500 Kbps. Given theses parameters, even streaming from a single sender would likely produce great results without even challenging the system. There was also an absence of an in-depth analysis of the scheduling algorithm, i.e., there was no mention of the effects of streaming bit rate and different number of senders on the algorithm performance.

It was observed that after 700 seconds, the missed segment ratio for PGA and DNC was estimated to be the same and thus once the system achieves stability, DNC no longer provides an advantage over PGA. In addition, experimental results showed that the buffering delay of DNC is only a slight improvement over PGA. The buffering delay of DNC is approximated to be between 1 to 2 seconds, whereas PGA has a buffering delay of approximately 3 to 4 seconds (after system warm up).

The authors of [48] describe a receiver-driven streaming component that works with existing media streaming clients. This Local Proxy Stream Server (LPSS) is a commercial product installed on the client machine that handles the multi-sender streaming while the client streams the received content to the local media player. It implements a tracker that manages all other peers currently streaming the same content. This tracker maintains the global state about all the nodes in the system and can store entire media files.

This approach uses a Block Scheduling Algorithm approach where bandwidth estimates are computed using synchronous exponential averaging. The streaming starts with a Round-Robin assignment, which assumes that every sender has the same bandwidth, and the scheduling algorithm attempts to minimize the request load on servers. Also, the scheduling algorithm follows an earliest deadline first pattern until it finds a potential miss (based on the bandwidth and load estimates, and block deadlines). When that happens, it attempts to perform a connection swap, which means reassigning the currently unassigned segment to another sender at an earlier time and taking the other segment from that peer. This is an NP-Complete problem but it can be approximated using a simple heuristic approach. If the heuristic approach fails, a block splitting strategy is adopted where a large block stuck with a slow connection is divided into smaller blocks, which can be downloaded in parallel. This may help in some situations but it also results in a higher HTTP overhead and thus should only be used as a last resort. Block splitting techniques are performed recursively until a feasible assignment is found.

The positive aspects of this work are that HTTP is used as transfer protocol, which makes it easier for integration with the existing infrastructure. Unlike web servers, however, no media streaming peer system is as widely available in a wide-area setting. A practical approach has been taken here by using the existing media servers over HTTP and RTP. Also, real prototype implementation, deployment, and experiments have been carried out using PlanetLab. The experiments carried out show the effects of different number of senders and different video bit rates but they are not reflected upon by using standard metrics such as

continuity index.

The main negative aspect of this solution is that since its protocol uses HTTP for transferring data, it incurs a lot of overhead. This overhead may be reduced if the block size is increased but larger block sizes result in reduced efficiency of scheduling. Also, only CBR videos are considered and no comparison has been made to other scheduling algorithms. There has only been one comparison and that has been with BitTorrent [5], which is clearly not a stream scheduling algorithm. The block scheduling algorithm used is not new; it reflects an earliest deadline first approach with a few additions such as connection swapping and block splitting.

The above works, although most significant, were not the only approaches to the scheduling problem and many other solutions have been proposed. Some interesting algorithms, such as [69], split the video streams into quality layers and ensure that the scheduling algorithm is aware of the quality layers so that it can give the base layers a higher priority to provide a minimum quality for most users. Peers with sufficient bandwidth have the option of taking advantage of the higher quality streaming layers. Some scheduling algorithms perform a deeper analysis of the video stream and focus on giving priority to frames, which (if missed) would case the most distortion to the resulting video [58]. This algorithm increases the priority of such critical frames when creating the segment schedule. Other works were less than creative in their proposed solutions such as [6] where efficient scheduling of video segments is simply ignored. Instead, a naive algorithm is proposed where the fundamental concept is to simply calculate the time required to download the entire video from a number of nodes and use it to set the delay. Many solutions mainly address the overlay construction problem but fail to describe an efficient scheduling algorithm. Finally, some works are unclear in the description and analysis of the scheduling details such as segment assignment, CBR or VBR video type, and fail to mention any simulation or prototype results [65].

Chapter 3

Analysis of Existing Scheduling Algorithms

In this chapter we analyze current scheduling algorithms, namely Round-Robin, Random, and Rarest First that we use as a basis of comparison against the ODV scheduling algorithm. We also introduce schedule trace graphs in Section 3.2, which are tools for visually representing the results of a streaming session. We then use the schedule trace graphs to describe the behavior of the current schedule algorithms in more detail in Sections 3.3, 3.4 and 3.5.

3.1 Introduction

During a video streaming session, the receiving peer receives video segments from a subset of active peers in the network. Before the actual streaming session can start, a video segment assignment, also called a schedule, must be created so that each sender knows which segments it is responsible for transmitting. Although the master schedule, created by the receiver, contains the schedules for all senders, individual senders only need to know their own schedules. An efficient segment assignment is one of the major factors determining the quality of the video stream and in this section we describe several existing scheduling algorithms that will be used as a basis for comparison with our proposed scheduling algorithm. These algorithms include the Round-Robin, Random, and Rarest First.

The Round-Robin and Random algorithms were chosen because many of existing P2P

streaming systems and prototypes still use these naive algorithms. Such systems tend to focus more on overlay construction and content availability broadcasts than on the scheduling problem and these algorithms are the easiest to implement. They serve very well to establish a basis for comparison against the more advanced scheduling algorithms.

The Rarest First algorithm was chosen because it is the scheduling algorithm of the CoolStreaming/DONet [71], one of the first P2P streaming system to attract over 1 million clients. This scheduling algorithm is still in use, as part of CoolStreaming, as the base technology for live IPTV programs launched by Roxbeam Corporation jointly with Yahoo Japan, in October 2006. It is very useful to see how our algorithm compares against Rarest First, a sophisticated scheduling algorithm that has been deployed in a real life deployment scenario for several years.

The advanced scheduling algorithms, such as Rarest First and ODV, utilize the bandwidth estimates from each sender to the receiver during the schedule creation. In networks such as the Internet where bandwidths are dynamic and heterogeneous, the frequency of sending updated schedules is clearly a trade off. Sending schedules at lower frequency results in a slower adaptation to the changing network conditions, while sending them at higher frequency introduces higher communication and computational overhead. For most practical considerations, this computational overhead is negligible for all algorithms presented in this thesis. See Chapter 5 for details.

As mentioned in Chapter 2, most of the current scheduling algorithms estimate the size of each segment by assuming a constant bit rate. This is quite accurate for the CBR videos but in case of VBR videos can lead to inefficient schedules, missed segment deadlines, and ultimately lower quality of the playout. To the best of our knowledge, the ODV is the only algorithm that takes the bit rate variability into consideration when creating a schedule. See section 4.2 for more details.

The rest of this chapter describes the various scheduling algorithms, which we use to compare against our ODV algorithm, as well as the tools used for this comparison. One of the most effective qualitative tools are the schedule trace graphs, described in section 3.2. Also, to further demonstrate each algorithm, we present a sample scenario of scheduling a 10-segment video clip. In this scenario, we arbitrarily select typical streaming system values to merely illustrate some major characteristics of each scheduling algorithm. We assume that each segment has a constant size of 50 KB and the average video bit rate is 400 kbps. We also have 3 senders, with respective bandwidths of 400 kbps, 240 kbps, and 160 kbps.

Since each segment has a constant length of 50 KB, these senders take 1.25 s, 1.67 s, and 2.5 s respectively, to transmit one segment. Finally, since the average bit rate of the video is 400 kbps, we can estimate that each 50 KB segment contains 1 second of video.

Based on these assumptions, sections 3.3, 3.4, and 3.5 describe the process that each algorithm uses to create the schedule for this scenario, and we discuss the strengths and the weaknesses of each approach. We also show actual streaming traces to further illustrate the strengths and weaknesses of each algorithm.

3.2 Schedule Trace Graphs

When comparing various scheduling algorithms, it is important to gain deeper insight into the details of their operation at the lower level so that we can understand and predict their resulting behavior at a more macroscopic scale. In our case, it is extremely useful to be able to examine the transmission of every video frame from the senders to the receiver. As detailed in Chapter 5, we collect a vast amount of data pertaining to the scheduling algorithm operation. For a more intuitive analysis, we generate graphical representations of all streaming sessions, which we call schedule trace graphs or streaming traces. We use these schedule trace graphs to demonstrate and explain the real-life effects of the algorithms on the video frame delivery. In this section, we explain how to read these diagrams since they are used in later chapters to describe the strengths and weaknesses of each scheduling algorithm. Figure 3.1 shows a sample schedule trace graph obtained from the receiving peer as a result of a 20-segment schedule using the Random algorithm.

The y-axis represents the video time and the x-axis represents the data offset within the video file. The video playout is represented by the line that traverses diagonally from the lower-left corner of the graph to the upper-right. If we had used CBR videos, the playout would be represented by a straight line with a slope that was an inverse of the video bit rate. In all of our experiments, however, we use VBR videos and as such the slope of the playout line is usually not constant. In general, steeper slope represents lower bit rate portions of the video, whereas flatter slope represents higher bit rate portions. Most schedule traces that we will examine have playout lines with a varying slope.

Note that the schedule trace graph also contains multiple, short horizontal lines. These lines are actually composed of data points that represent individual video frames. Each video frame data point indicates the sequence of the particular video frame and its arrival time

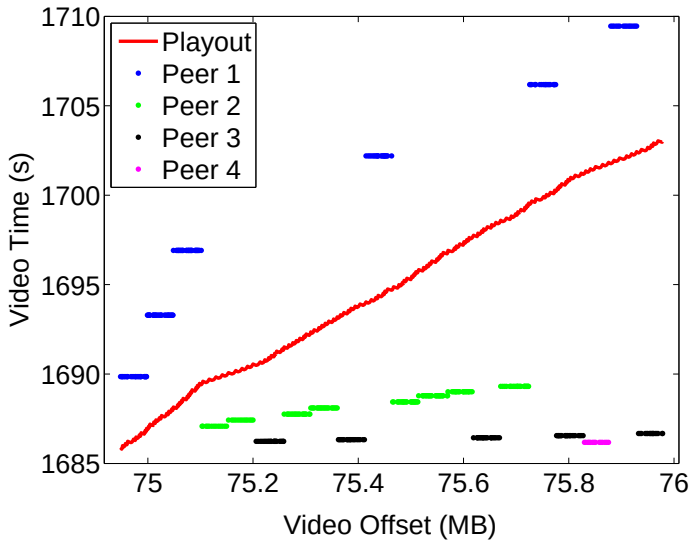


Figure 3.1: Sample random algorithm streaming trace obtained from PlanetLab experiments.

at the receiving peer. Since the sender peers transmit segments composed of multiple video frames, we see groups of frames arriving at the same time, forming the short horizontal lines. Finally, note that the frames delivered from different sender peers have different colours.

The most important pieces of information represented by the schedule trace graphs are the relative locations of the received video frames and the playout line. All frames below the playout line have met their deadlines whereas all frames above the playout line missed their deadlines. Note that this does not take the buffering delay into consideration. Buffering delay would essentially move the playout line higher on the graph, ideally into a position just above all the received video frames. In our experiments, we assume no buffering delay as we are more interested in the relative performance of the scheduling algorithms and not the absolute values of the continuity index.

In the sample trace graph in Figure 3.1, we can see that sender peer 1 is slow and all of its transmitted video frames have missed their deadlines. Sender peers 2, 3, and 4 are faster and all their video frames have made their deadlines. We can clearly see that schedule trace graphs are effective tools for presenting and analyzing the behavior of scheduling algorithms, because they provide visual feedback of the streaming session characteristics.

Peer BW/Segment	1	2	3	4	5	6	7	8	9	10
400 kbps	1.25 s			2.50 s			3.75 s			5.00 s
240 kbps		1.67 s			3.33 s			5.00 s		
160 kbps			2.50 s			5.00 s			7.50 s	

Table 3.1: Sample round-robin algorithm schedule.

3.3 Round-Robin Algorithm

Round-Robin is most likely the simplest scheduling algorithm. When creating a schedule, it assigns every consecutive segment to the next sender in a sender list. Once we reach the end of the sender list, we start from the first sender again. We continue in this manner until all segments have been assigned to the candidate senders. Every sender will have the same number of segments assigned and no preference is given to any peer based on the bandwidth estimate, reliability, or any other factors.

Simple Round-Robin schedulers, which assign equal portions of responsibility to all participants, are still widely deployed for their simplicity but are not suitable for Internet streaming applications because of bandwidth heterogeneity. They are suitable, however, to serve as a basis for comparison against more sophisticated scheduling algorithms such as Rarest First and ODV. The obvious advantage of this algorithm is that it is very simple and fast. Its main disadvantage is the fact that it does not take the network conditions into consideration and that usually results in inefficient schedules.

We now describe the Round-Robin solution to the sample scheduling scenario introduced in section 3.1. First off, no calculations are necessary and all segments are evenly spread among all candidate sender peers. The resulting schedule is shown in table 3.1. All entries in the table indicate the expected arrival time of each segment, which is based on the sender bandwidth estimates.

We see that peer 1 (400 kbps) is assigned segments 1, 4, 7, and 10, peer 2 (240 kbps) is assigned segments 2, 5, and 8, and peer 3 (160 kbps) is assigned segments 3, 6, and 9. The advantage of this algorithm is that we achieve the highest degree of load balancing among the peers, i.e., all peers are assigned an equal number of segments. Also, this degree of load balancing is always guaranteed. The disadvantage here is that although all peers are assigned equal number of segments, the faster peers that can handle a higher load will tend to be underutilized whereas the slower peers can be over-utilized. This usually leads to missed deadlines on segments assigned to the slower peers since they may not have the

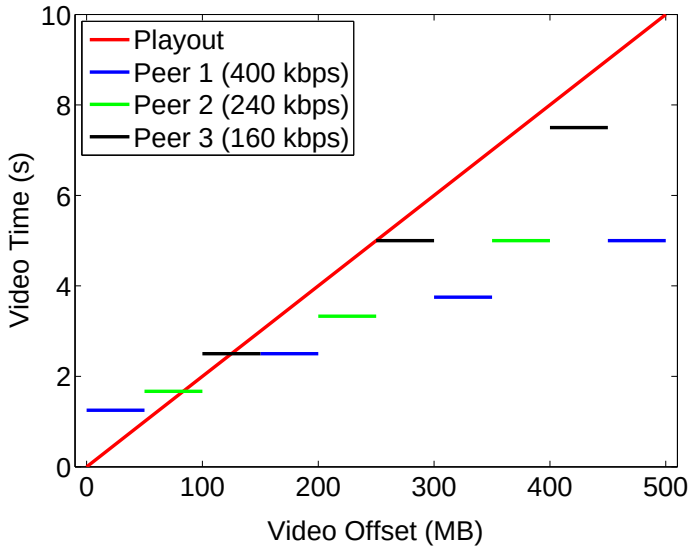
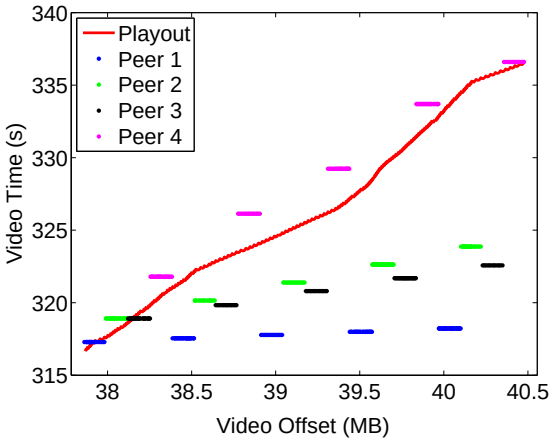


Figure 3.2: Streaming trace for the sample round-robin schedule.

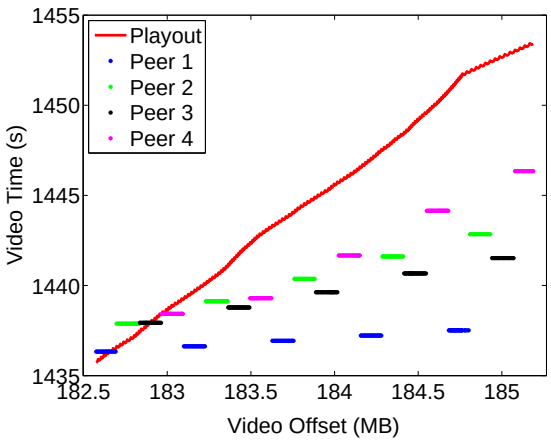
capability to handle the assigned/expected load. Figure 3.2 shows the schedule trace graph for this sample schedule. We can see that in this example, the Round-Robin algorithm does quite well as most of the video frames are below the playout line, thus meeting their deadlines.

Let us now examine some streaming traces from PlanetLab experiments. Figure 3.3 shows a couple examples of the Round-Robin algorithm in action. Figures 3.3(a) and 3.3(b) show successful schedules from 4-sender traces. In Figure 3.3(a), although most frames sent by peer 4 miss their deadlines, we can see that a small buffering delay would solve this problem. Peer 4, however, is the slowest peer and a more efficient scheduling algorithm would not assign it as many segments. Figure 3.3(b) shows that the Round-Robin algorithm is capable of producing efficient schedules. Only few frames miss their deadlines in this example and a very small buffering delay would fully compensate for this.

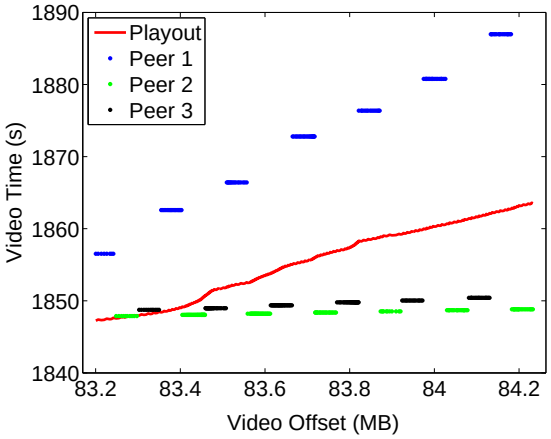
Figures 3.3(c) and 3.3(d) show failed schedules from 3 and 5-sender sessions. Figure 3.3(c) shows a 3-sender trace where the indiscriminate assignment to senders without considering the bandwidth results in 1/3 of the segments missing their deadlines. In order to compensate for all the late segments delivered by peer 1, we would need to use a large buffering delay. Also, if we consider the pattern of the frames sent by peer 1 and compare it to the playout line, we see that they have different shapes (or create lines with different



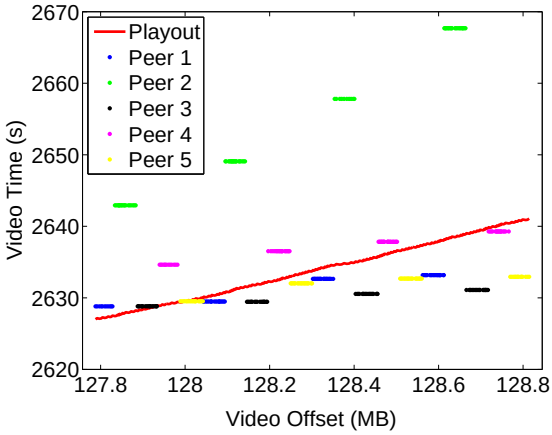
(a) Successful Round-Robin schedule.



(b) Successful Round-Robin schedule.



(c) Failed Round-Robin schedule.



(d) Failed Round-Robin schedule.

Figure 3.3: Round-Robin algorithm streaming traces obtained from PlanetLab experiments.

Peer/Segment	1	2	3	4	5	6	7	8	9	10
400 kbps	1.25 s					2.50 s				
240 kbps			1.67 s					3.33 s	5.00 s	
160 kbps		2.50 s		5.00 s	7.50 s		10.00 s			12.50 s

Table 3.2: Sample random algorithm schedule.

slopes). As such, we would need to increase the buffering delay by a larger amount that would move the end of the playout line over the last segment sent by peer 1. Figure 3.3(d) shows a very similar pattern to Figure 3.3(c) except that in this case, we are dealing with 5 senders and two of them are unable to deliver their frames on time. Once again, a more efficient scheduling algorithm would avoid assigning segments to those two peers.

3.4 Random Algorithm

A perfect example of the Random scheduling algorithm is the Chainsaw scheduling algorithm [47]. In this scheduling algorithm, every segment is assigned to a random sender from the sender list. Similarly to the Round-Robin algorithm, no preference is given to any sender based on bandwidth or reliability estimates. In most implementations, the random seed is based on system time, resulting in pseudo-random schedule. This algorithm is very simple and much like the Round-Robin algorithm, it serves as a basis for comparison against other scheduling algorithms.

The advantage of this algorithm is that there is a chance of an efficient segment assignment and the changing network conditions usually do not affect the performance of the algorithm, making it rather stable. The main disadvantage is that the probability that the resulting assignment will be anywhere close to an optimal schedule is relatively low.

We now describe the Random algorithm solution for the sample scheduling scenario introduced in Section 3.1. In the example schedule shown in Table 3.2, the slowest peer is assigned the most segments, which results in a very inefficient schedule. This is further confirmed by the schedule trace graph for the sample schedule shown in Figure 3.4. We can clearly see that all segments from peer 3 are above the red playout line and thus miss their deadlines.

Let us now examine some streaming traces from PlanetLab experiments. Figure 3.5 shows some examples of the random algorithm in action. In Figures 3.5(a) and 3.5(b) we can see that it is possible for a random algorithm to come up with an efficient schedule. In

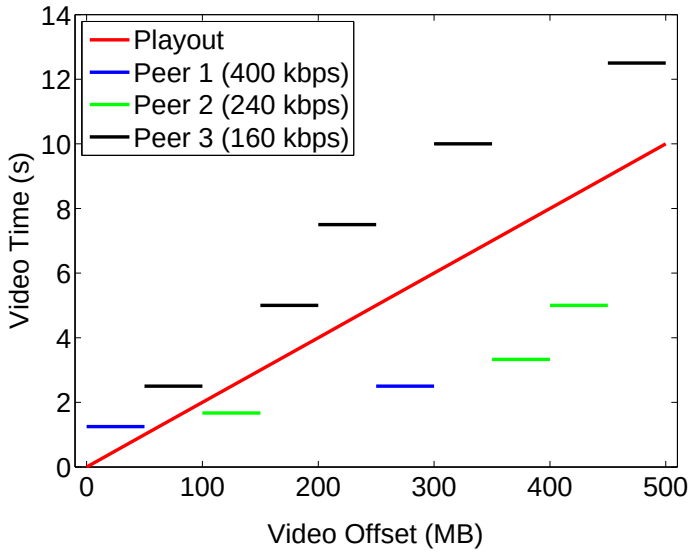
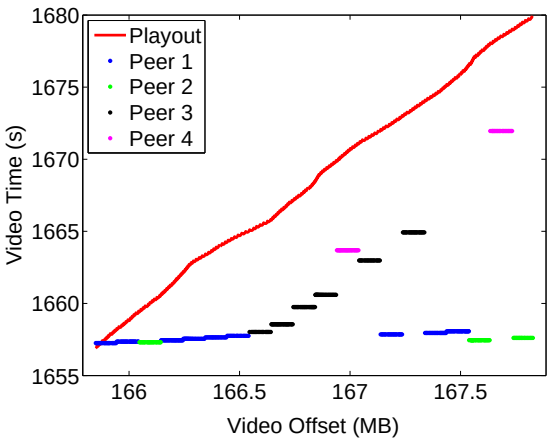


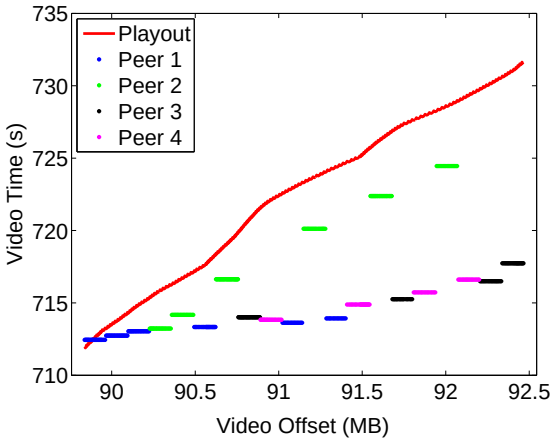
Figure 3.4: Streaming trace for the sample random schedule.

Figure 3.5(a), we have 2 very fast senders (peers 1 and 2), 1 fast sender (peer 3), and 1 slow sender (peer 4). The segment assignment is such that that peers 1 and 2 get most of the early segments, peer 3 gets some segments in the middle of the trace, and the slowest peer 4 only gets 2 segments near the end of the trace. This schedule is quite efficient for this session and all the segments meet their deadlines. Similarly, in Figure 3.5(b) we have an efficient schedule produced by the random algorithm. Peers 1, 3, and 4 are fast and they are assigned most of the segments. The slower peer, peer 2, just happens to get right segment assignment such that it is able to deliver all its segments on time.

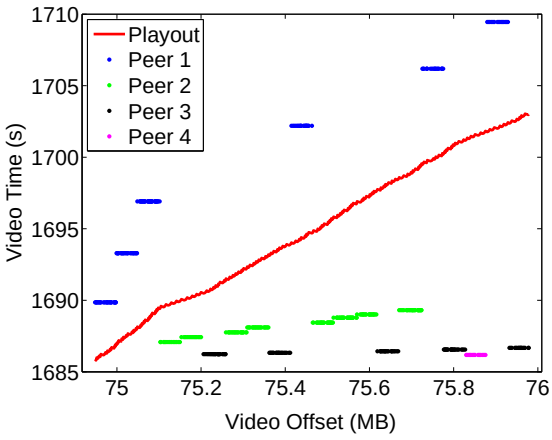
On the other hand, Figures 3.5(c) and 3.5(d) show inefficient schedules created by the random algorithm. In Figure 3.5(c), the random scheduler assigns most of the early segments to the slowest sender (peer 1), resulting in all those segments missing their deadlines. Figure 3.5(d) shows a trace where most of the segments are assigned to the slower peers (peers 1 and 4) where as the faster peers get only a few segments assigned. This clearly results in most of the segments missing their deadlines.



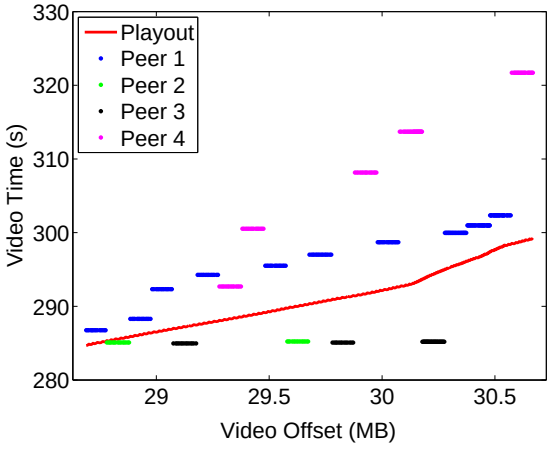
(a) Successful random algorithm schedule.



(b) Successful random algorithm schedule.



(c) Failed random algorithm schedule.



(d) Failed random algorithm schedule.

Figure 3.5: Random algorithm streaming traces obtained from PlanetLab experiments.

3.5 Rarest First Algorithm

The CoolStreaming algorithm is an implementation of the rarest first scheduling algorithm used by the CoolStreaming/DONet peer-to-peer system [71]. This heuristic algorithm first calculates the number of potential suppliers for each segment (i.e., the partners containing each segment in their buffers). The claim is that segments with fewer potential suppliers are more difficult to meet deadline constraints and as such the algorithm determines the supplier of each segment starting from those with only one potential supplier, then those with two, and so forth. In other words, the rarest segments get scheduled first. If there is a segment with multiple potential suppliers, the supplier with the highest bandwidth is selected [71].

The main advantage of this algorithm is that it tends to assign more segments to the fastest peers, which also tend to be the most reliable peers. The disadvantage is that in many cases this scheduler will reduce to streaming from a single host even though there may be multiple partners available. In large scale systems, this may lead to overloading of the faster peers. Also, this algorithm relies on the absolute values of the bandwidth estimates when determining whether a sender has enough available time to transmit a segment. Since bandwidth estimates are rarely 100% accurate, algorithms that rely on relative ratios of the bandwidth estimates instead of absolute values tend to achieve better performance and create better load sharing schedules. Another problem with relying too much on the absolute bandwidth estimate is that abrupt changes in bandwidth or congestion of the connections with the fastest peers may result in many segments missing their deadlines.

Let us now describe the Rarest First algorithm solution to the sample scheduling scenario introduced in Section 3.1. The schedule produced by the Rarest First algorithm for our sample scenario is shown in Table 3.3. We can see that the resulting schedule is not fully balanced because the fastest peer is assigned the most segments, while the slowest peer is assigned nothing. A more efficient algorithm would at least try to utilize the slowest peer since it most cases it could efficiently deliver at least one segment. Figure 3.6 shows the resulting streaming trace for this sample scenario. We can clearly see that this schedule is not optimal as many segments are close to missing their deadlines. Although a small buffering delay would take care of this problem, the point is that the algorithm did not find a schedule that allows for as much network fluctuations as possible and if anything changes with the peer whose bandwidth estimate was the fastest at the time of scheduling, there

Peer/Segment	1	2	3	4	5	6	7	8	9	10
400 kbps	1.25 s		2.50 s	3.75 s	5.00 s		6.25 s	7.50 s	8.75 s	10.00 s
240 kbps		1.67 s				3.33 s				
160 kbps										

Table 3.3: Sample rarest first algorithm schedule.

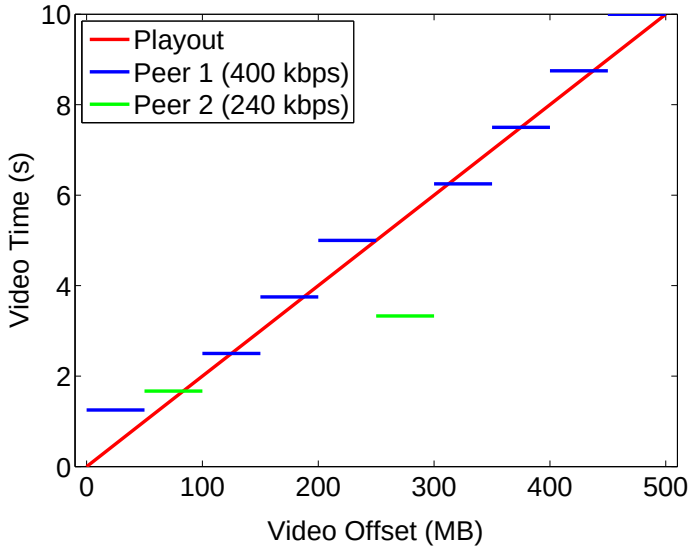
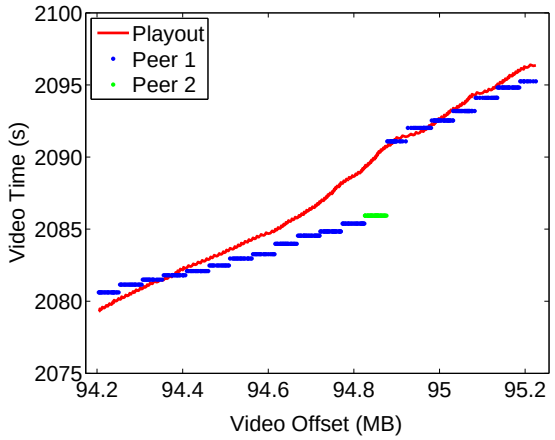


Figure 3.6: Streaming trace for the sample rarest first schedule.

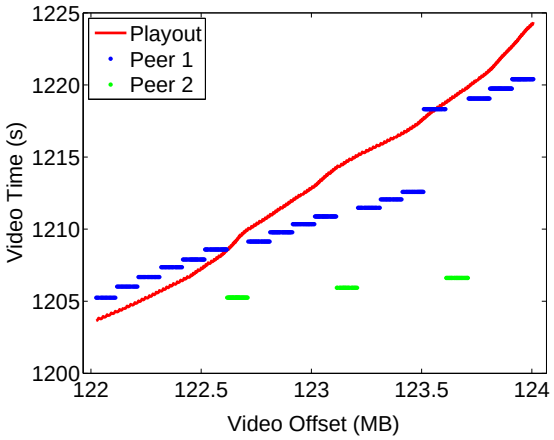
is good possibility that most segments will miss their deadlines. Basically, although the segment assignment seems efficient, it does not leave much room for error.

Let us now examine some streaming traces from PlanetLab experiments. Figure 3.7 shows several examples of the rarest first algorithm in action. In Figures 3.7(a) and 3.7(b) we have two examples of efficient schedules created by the rarest first algorithm. It is important to note, however, that in both those cases, there were 5 potential senders available, yet the scheduling algorithm mostly used only one of them. As mentioned before and as evidenced from the Rarest First solution to our sample scenario, this is one of the characteristics of this algorithm.

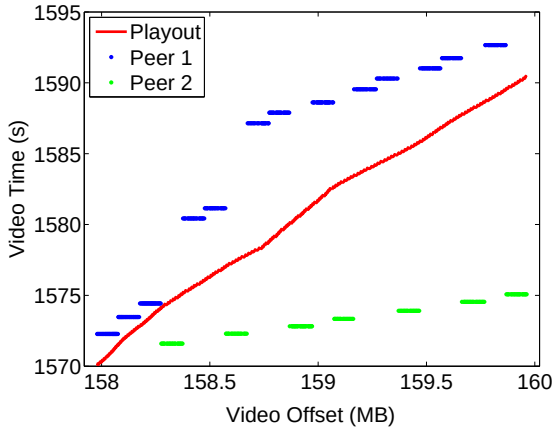
Network fluctuations are common in the Internet and sometimes the link with another peer slows down and/or gets delayed. Figures 3.7(c) and 3.7(d) show exactly such occurrences. In both cases, at the time of scheduling, Peer 1 was the fastest sender and most segments got assigned to it. At run time, for whatever reason, the network communication



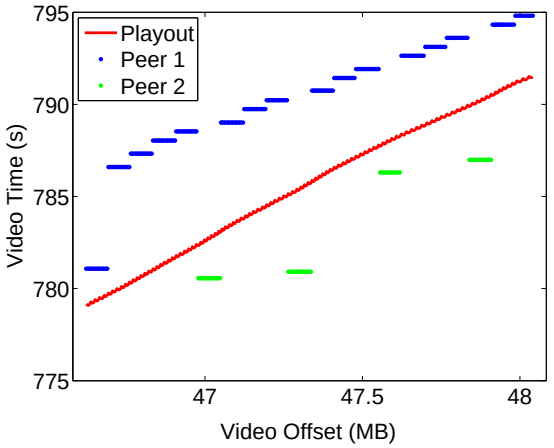
(a) Successful rarest first algorithm schedule.



(b) Successful rarest first algorithm schedule.



(c) Failed rarest first algorithm schedule.



(d) Failed rarest first algorithm schedule.

Figure 3.7: Rarest first algorithm streaming traces obtained from PlanetLab experiments.

gets delayed and as a result there are many segments that miss their deadlines. The lesson for other scheduling algorithms that can be drawn from these failed schedules is that we should not rely too much on a single peer in a streaming session and instead try to utilize all available resources to an appropriate degree. This is one of the characteristic of our ODV scheduling algorithm described in Chapter 4.

Chapter 4

Proposed Scheduling Algorithm

In this chapter, we introduce our proposed scheduling algorithm, ODV. In Section 4.2 we describe how ODV provides VBR-specific handling of video data. Section 4.3 defines the ODV algorithm itself and schedule trace graphs are used to provide more detail insight into its behavior. We conclude in Section 4.4 by proving the polynomial bounds for space and time complexity of the ODV algorithm execution.

4.1 Introduction

The primary objective of a scheduling algorithm is to create a schedule such that all requested video segments are delivered to the receiver before their respective deadlines. If that is impossible to achieve (given insufficient resources), we want to minimize the number of segments missed and minimize the buffering delay required. We also have two secondary objectives. We want to make efficient use of the bandwidth and avoid transmitting segments before they are required because there is no guarantee that they will even be used. Finally, since we do not want to overload any of the senders, we need to make sure that the load is balanced across all available resources.

The proposed On-time Delivery of Variable bit rate (ODV) algorithm assigns each segment to a particular sender at a specific transmission time slot. Unlike in other scheduling algorithms, under the ODV algorithm the sender for each particular segment is selected partially based on the current estimated load of that sender. The estimated load of a sender is the amount of work (time spent transmitting segments) that the sender has already been assigned by the schedule. The amount of work that a particular sender will need to perform

in order to transmit a segment depends on the segment size and the sender's bandwidth. We predict the load of each sender by temporarily assigning the next scheduled segment to each sender and we select the peer with the smallest predicted estimated load as the sender. This ensures that the load is shared among all senders.

In situations where senders do not have all segments available, we must make sure that we do not commit a sender who is the only one that can deliver particular segments to do work delivering other segments, which could be delivered by other senders. Therefore, when determining the appropriate peer to deliver a segment, we must also consider the number of potential senders for that particular segment. It is usually more difficult to meet the deadline of a segment that has fewer potential suppliers. For this reason, the algorithm first calculates the number of potential suppliers for each segment and the segments with least potential suppliers are scheduled first. That is, the segments with one potential supplier are scheduled first, then segments with two potential suppliers, then three and so on. This part of the algorithm is similar to the Rarest First algorithm but this is a potential area of improving the algorithm in the future because it is not always necessary to schedule the segments with the least potential suppliers first. It is possible to change the segment scheduling order (from the order of the number of potential suppliers) and still create an efficient schedule where all segment deadlines are met.

Another characteristic that improves the ODV scheduling algorithm over all others is the support for VBR videos. The details of this characteristic are described in the next section.

4.2 VBR Support

We have not seen an algorithm that directly claims to support VBR videos. Previous works that even include this level of detail, merely estimate the data length of a constant duration segment by using the average bit rate of the entire video. In order to be able to accurately calculate the data length of each segment, the scheduler would need to have access to the video trace information, i.e., the metadata describing the frame and segment sizes for a particular video. This introduces the problem of distributing this trace information to the potential receivers, however, distributing this trace information to peers is not difficult because the trace data is very compact compared to the video itself. For example, the length of trace data can be compacted to 10 Bytes per video frame whereas the frame length can

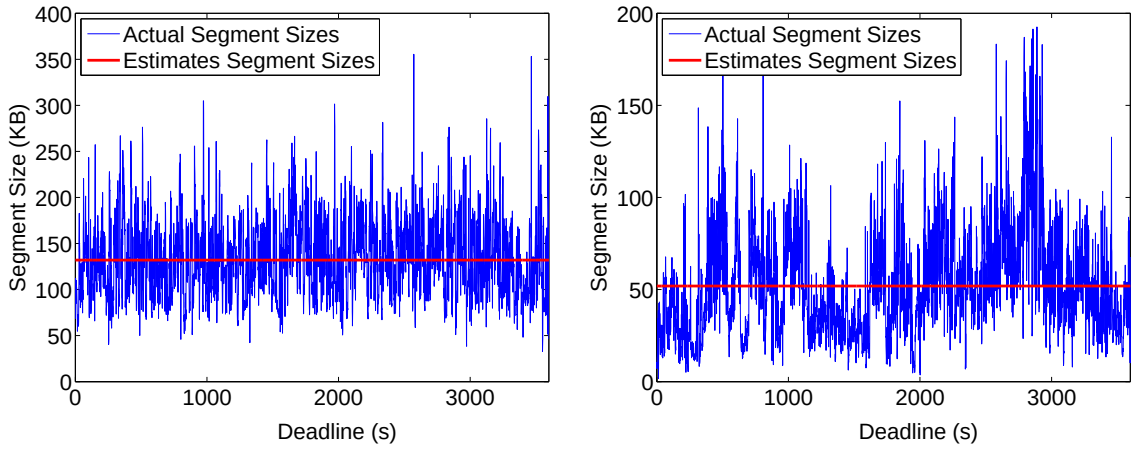
vary from 100 Bytes (for low quality videos) to 12 KB (for high quality videos). So the trace data for 100 frames would require 1 KB. Depending on segment availability, it would make it rather efficient to piggy back the trace information with data packets. To further compact the trace information, instead of sending the frame information to the receiver, we could simply ask the senders for segment-level trace information. For example, we would like to know the segment sizes for 120 1-second segments starting at some offset in a particular video. The response would be 120 times 2 bytes (a 2-byte field is sufficient to describe the segment data length) resulting in 240 bytes of trace information for 2 minutes of video.

One option could be to store the trace files in dedicated servers or peers and send requests to those servers/peers to create a schedule for a particular scenario. This would add some overhead because the receiving peer would need to outsource its scheduling.

Another option could be to still store the trace files in dedicated servers or peers but we would not send scheduling requests to those machines. Instead we would simply request the trace information for a portion of the VBR video. For example, a 5-minute long VBR video would not require a great amount of bandwidth to transmit so this approach could be practically feasible. This way, the receiving peer would have all the information required to create an efficient schedule for a VBR video.

The problem introduced in both of these scenarios is the need for distributing the trace information or a way of transmitting accurate deadline estimates. This increases the complexity of the system but also this could be an optional component. If available, the scheduler would use the trace information to create more accurate schedules for VBR videos; otherwise, it would default to using segment deadline estimates based on average video bit rates. As we will later see, however, using the trace information during the scheduling results in more accurate schedules.

Consider Figure 4.1 that shows the graphs of 1-second segment sizes for 2 high quality videos used in the experiments. The red line indicates the average bit rate estimate for the entire video whereas the blue line indicates the actual segment sizes for the 1-second video. In all instances where the actual segment sizes are larger than the estimated ones, the risk of inefficient scheduling increases. This is because we could assume that, for example, each 1-second segment has the size of 55 KB whereas the actual sizes of 1-second segments for this particular video portion vary between 70 and 80 KB. In some cases, as evidenced by the graphs, the difference between the actual and the estimated segment sizes could be as high as 400%. So even with accurate bandwidth estimates, the segment assignments would



(a) High Quality Soccer VBR Video With 1-Second Segments.

(b) High Quality Alladin Cartoon VBR Video With 1-Second Segments.

Figure 4.1: Variability of segment sizes in video streams.

likely lead to missed deadlines since the segment sizes were underestimated.

4.3 ODV Algorithm

A high-level pseudo code of the proposed scheduling algorithm is shown in Fig. 4.2. This algorithm is executed by the receiver in a streaming session. As mentioned before, the receiver collects segment availability information from potential senders, which is usually obtained by exchanging buffer maps among peers. In addition to segment availability, the buffer maps include the size of each segment. The segment size can easily be computed by parsing the headers of the encoded video stream. Alternatively, the segment sizes could also be stored in a small meta file associated with the video. We notice that including the segment size in the buffer maps adds negligible overhead compared to the video traffic. For example, current scheduling algorithms use at least one field to indicate the availability of each segment. In our algorithm, we use the same field, but include the size of the segment if it is available and zero otherwise. A 2-byte field is sufficient to store the size of the segment in our algorithm. For a scheduling window of 30 segments, i.e., 30 seconds of video data, segment sizes need only 60 bytes which is indeed negligible compared to the video data that is in order of, at least, several kilo bytes. The ODV algorithm also estimates the bandwidth of each peer based on the history of the connection between the receiver and that peer.

After collecting the needed information for a scheduling window, the algorithm computes the expected delivery time t_d for each segment s when it is assigned to each one of the potential senders p. Then it chooses sender p_s that will deliver segment s at the earliest delivery time t_{ed}. In this assignment, the algorithm begins with the segment that has the fewest number of potential senders, similar to the Rarest First algorithm. However, unlike the Rarest First algorithm, the ODV algorithm does not simply select the sender with the highest bandwidth and enough available time. Rather, the ODV algorithm considers the current utilization of each peer, util(p), to estimate the expected delivery time of each segment and achieve the ultimate target of on-time delivery of video data.

The ODV algorithm takes a couple ideas from the Rarest First algorithm but it also introduces several changes, which as we will see in Chapter 5 results in better performance. Much like Rarest First, the ODV algorithm schedules segments with fewest potential suppliers first but when selecting a sender from a group of potential suppliers, it does not simply select the sender with the highest bandwidth and enough available time. Among multiple potential suppliers, the sender that can deliver a segment at the earliest time is selected. Thus, the ODV algorithm may actually assign segments to slower peers before fully saturating faster peers, if the slower peers will deliver them earlier. This algorithm thus benefits from the Rarest First algorithm's idea of giving preference to the faster senders while it also achieves better resource sharing by utilizing the slower senders.

Let us now work through an example of how the ODV algorithm creates a schedule. In our sample scenario, introduced in Section 3.1, we have 3 sender peers with respective bandwidths of 400, 240, and 160 kbps. Segment 1 is assigned to peer 1 because it can deliver it within 1.25 sec. Segment 2 could be delivered by peer 1 at 2.5 sec, 1.67 sec by peer 2, and at 2.5 sec by peer 3 - since peer 2 can deliver it at the earliest time, it gets assigned segment 2. Segment 3 could be delivered by peer 1 at 2.5 sec, peer 2 at 3.33 sec, or peer 3 at 2.5 sec. Both peer 1 and peer 3 could deliver segment 3 at 2.5 sec but peer 1 has a higher bandwidth estimate so it gets assigned segment 3. We continue to process all 10 segments in this manner to arrive at the resulting schedule shown in Table 4.1. The schedule trace graph for this scenario is shown in Figure 4.3 and if we compare it to the graphs for the other algorithms, we see that the ODV algorithm creates most space between the segment delivery and the playout line.

We now examine the streaming traces, shown in Figure 4.4, from the PlanetLab experiments to illustrate the real-life behavior of the ODV algorithm. In Figures 4.4(a), 4.4(b)

ODV: On-time Delivery of VBR streams

```
1.   for n = 1 to senderCount
2.     foreach s in potentialSegments[n]
3.     // segment s with n potential suppliers
4.       p_s = ∅ // selected sender
5.       t_ed = N_max // earliest delivery time
6.       foreach p in suppliers[s]
7.       // potential senders p for segment s sorted by bandwidth
8.           t_x = size(s) / bandwidth(p) // tx time
9.           t_d = util(p) + t_x // delivery time
10.          if (t_d < t_ed)
11.            p_s = p
12.            t_ed = t_d
13.          end if
14.        end for
15.        util(p_s) = util(p_s) + (size(s) / bandwidth(p_s))
16.        assignSegment(s, p_s)
17.    end for
18.  end for
```

Figure 4.2: Proposed segment scheduling algorithm.

Peer/Segment	1	2	3	4	5	6	7	8	9	10
400 kbps	1.25 s		2.50 s			3.75 s	5.00 s			6.25 s
240 kbps		1.67 s			3.33 s			5.00 s		
160 kbps				2.50 s					5.00 s	

Table 4.1: Sample ODV algorithm schedule.

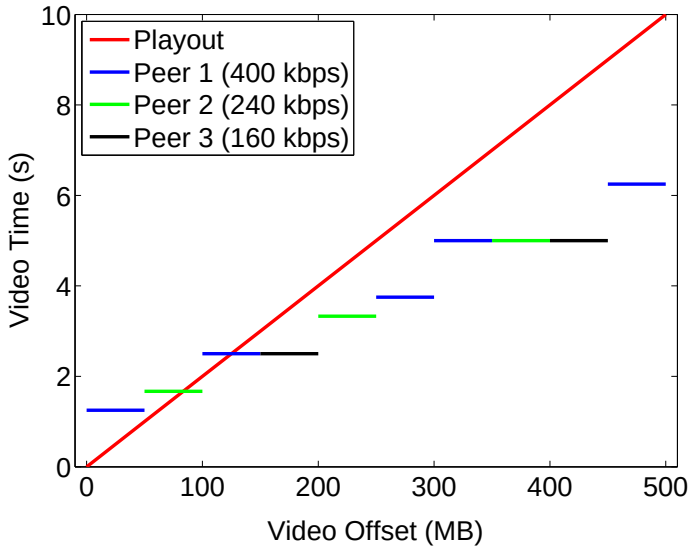
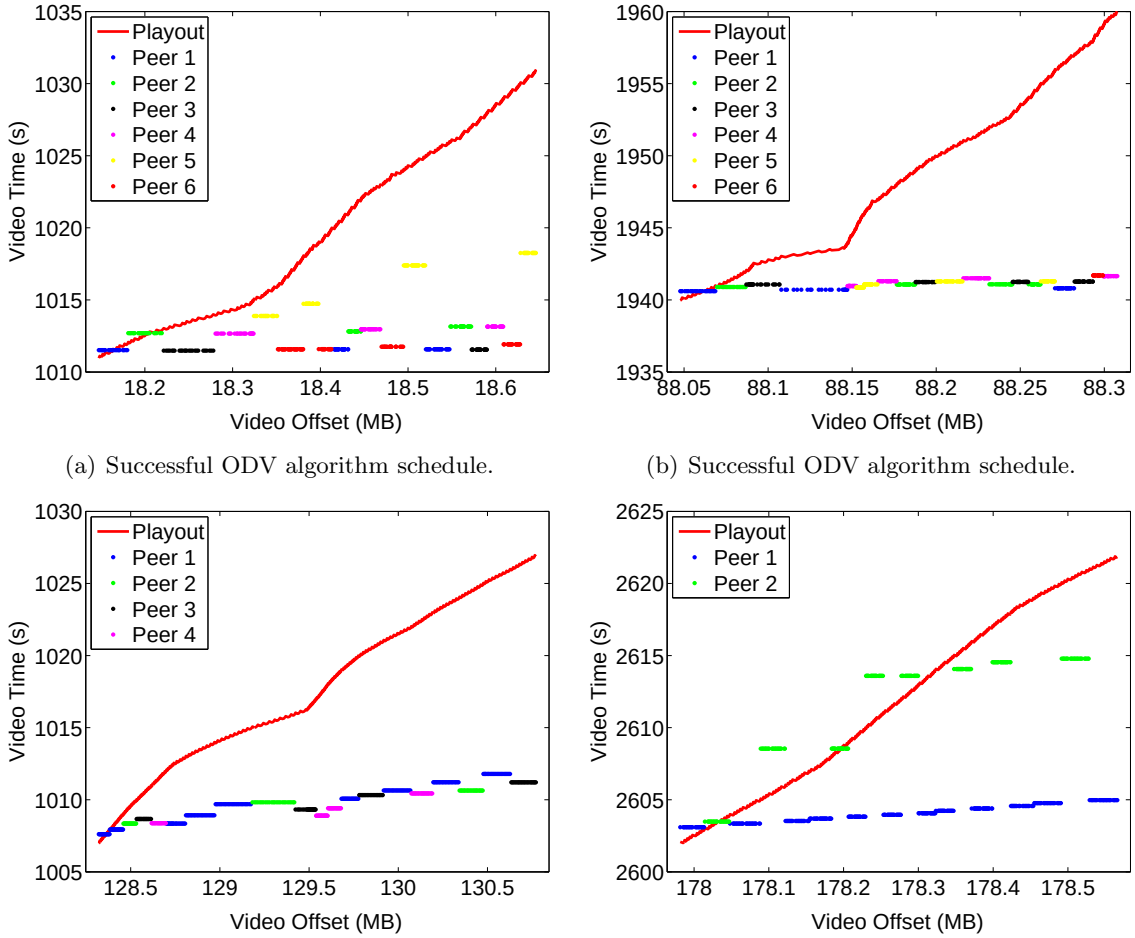


Figure 4.3: Streaming trace for the sample ODV schedule.

and 4.4(c) we have three examples of efficient schedules created by the ODV algorithm. The first thing we should notice is that the ODV algorithm tends to use most (if not all) of the sender peers when assigning segments. The fastest senders will still get most of the segments but the slower senders will be utilized much more than by the Rarest First algorithm. We should also notice that there is usually a relatively large room for error, as evidence by the time between the playout line and the arrived segments. Figure 4.4(d) shows an example of a failed schedule. Again, the communication with another peer can sometimes get slower and/or delayed and that is exactly what happened with peer 2 in this case. As a result there may be some segments that miss their deadlines. This can be handled, however, using different techniques such as buffering delay.

We should note another important difference between the ODV schedule traces and all other schedule trace graphs. In the case of the ODV scheduler, the segment size is variable whereas for all other scheduler, the segment size is uniform. The reason is that the ODV scheduler includes the VBR support as described in Section 4.2. It is possible to add the VBR support to other scheduling algorithms but in all our experiments, only the ODV algorithm had the advantage of using VBR support.



(a) Successful ODV algorithm schedule.

(b) Successful ODV algorithm schedule.

(c) Successful ODV algorithm schedule.

(d) Failed ODV algorithm schedule.

Figure 4.4: ODV streaming traces obtained from PlanetLab experiments.

4.4 Space and Time Complexity Analysis of the Algorithm

The following lemmas show that the ODV scheduling algorithm is efficient, i.e., it runs in polynomial time and space.

Lemma 4.4.1 *If the number of sender peers is given by p and the number of segments is given by s, then the time complexity of the ODV scheduling algorithm is bounded by $O(p^3 \cdot s)$.*

Proof: At a quick glance, the segment scheduling part of the algorithm contains 3 nested loops. The iteration over the sender counts (line 1) will visit p elements. The first nested iteration over the segments with a specific number of suppliers (line 2) will visit at most s elements. The sorting of the senders for each segment by bandwidth (line 6) will take at most $O(p \cdot \log p)$ (when using merge sort). Finally, we need to iterate over this sorted list of potential senders. This iteration will visit at most p elements. All other calculations occur in constant time. This gives us the following time complexity:

$$\begin{aligned} T &= O(p \cdot s \cdot (p \cdot \log p + p)) \\ &= O(p^2 \cdot s \cdot \log p) \\ &= O(p^3 \cdot s) \end{aligned} \tag{4.1}$$

Therefore, the ODV algorithm executes in polynomial time and is bounded by $O(p^3 \cdot s)$. □

Lemma 4.4.2 *If the number of sender peers is given by p and the number of segments is given by s, then the space complexity of the ODV scheduling algorithm is bounded by $O(p \cdot s)$.*

Proof: In Figure 4.2, the first task in the ODV algorithm that requires space is the iteration over the segments with n potential suppliers (line 2). Since $n \leq s$, we will need at most an s by p matrix for this data structure. We also know that we have p senders and thus there will be at most p schedules. Every schedule contains the list of segments assigned to that particular sender and their corresponding transmission start times. Since every schedule requires at most $O(s)$ space and we have p schedules, we need at most $O(p \cdot s)$ to store all schedules. Thus, aside from some constant temporary local storage, the total space requirement for the ODV scheduling algorithm is:

$$\begin{aligned} S &= O(p \cdot s + p \cdot s) \\ &= O(p \cdot s) \end{aligned} \tag{4.2}$$

Therefore, the ODV algorithm executes in polynomial space and is bounded by $O(p \cdot s)$. □

Chapter 5

Experimental Evaluation

In this chapter, we present our extensive evaluation and comparison of all segment scheduling algorithms examined in this thesis. We describe the experimental setup on the PlanetLab testbed in Section 5.2. Section 5.3 provides the algorithm comparison metrics such as continuity index, balance index and buffering delay while Section 5.4 displays the aggregated results obtained from the PlanetLab experiments.

5.1 Introduction

The experiments were performed on the PlanetLab testbed and the setup of the experiments is described in Section 5.2. As detailed in that section, diverse video streams were used with varying number of senders, varying location of receivers, and four different scheduling algorithms were evaluated. For each algorithm, more than 500 different experiments were conducted and the average results of these experiments are reported.

As mentioned earlier, most related work focuses on overlay construction and sender peer selection in P2P streaming systems, while neglecting the data scheduling aspect. This thesis focuses exclusively on the scheduling part of a P2P streaming system, and thus the goal of our experiments is to compare the performance of the scheduling algorithms presented in Chapter 3 and Chapter 4. Our experiments do not, however, cover overlay construction nor sender peer selection. The tasks of constructing the overlay network and the sender selection are performed manually during the experiment setup.

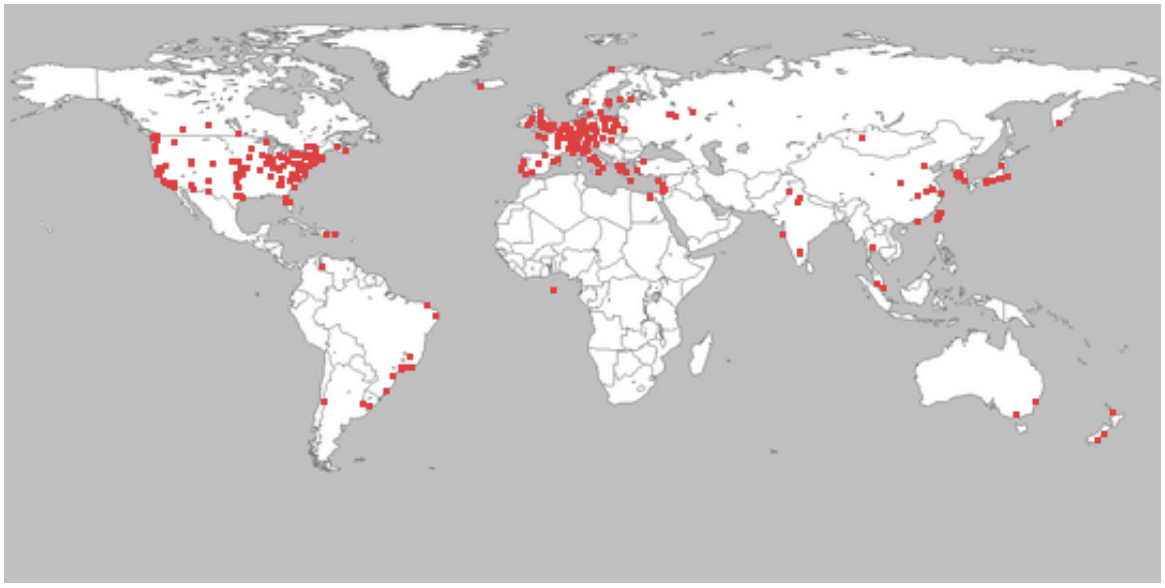


Figure 5.1: Global PlanetLab sites [49].

5.2 PlanetLab Setup

PlanetLab is a global research network created to support networking research. Since 2003, over 1000 researchers from industrial labs and academic institutions have used PlanetLab to develop new technologies for various applications such as distributed storage, network mapping, P2P systems, distributed hash tables, and query processing. As shown in Figure 5.2, PlanetLab currently consists of 1039 nodes at 487 sites. Each research project is given virtual machine access to a subset of the nodes [49].

Our experiments involved approximately 70 different PlanetLab nodes, which were uniformly spread throughout North and South Americas, Europe, Asia, and Australia. As shown in Figure 5.2, most of the PlanetLab nodes are located in North America and Europe. We only used a subset of all PlanetLab nodes to provide large geographical distances between each node in order to *stress* the scheduling algorithms. If we had used nodes very close to each other with abundant bandwidth, all algorithms would likely yield similar performance and we would not be able to uncover the unique characteristics of each algorithm that would surface in realistic resource-scarce environments. We segregated our PlanetLab nodes into 10 groups and we conducted a similar set of experiments within each node group.

We have developed a prototype P2P streaming system, in which we implemented all of the four segment scheduling algorithms analyzed in this thesis. Every active PlanetLab node ran a copy of the prototype system, which was capable of acting as a sender, a receiver, or both. We executed numerous tests to rigorously evaluate the performance of each algorithm. For every algorithm, we varied the number of senders from 2 to 6 and for each chosen number of senders, we used a diverse set of five variable bit rate (VBR) video streams, which we obtained from [63]. Table 5.1 summarizes various characteristics of these video streams and as the table shows, the chosen video streams have quite heterogeneous visual content, bit rate, motion, number of frames, and file sizes. This was done to create realistic evaluation scenarios. In total, we had 25 test cases with different videos and number of senders. Furthermore, every test case was repeated at 22 different PlanetLab nodes acting as receivers. Therefore, more than 500 experiments were conducted on PlanetLab to evaluate *each* of the four algorithms considered in this thesis. Every data point in the graphs shown in Figures 5.2 - 5.7 is an average of the 22 test cases mentioned above.

To manage such number of experiments, we designed an administrative interface to control all nodes from a central test driving application. The test driving application also controls numerous test parameters, including: receiver node, sender nodes, video stream, scheduling algorithm, segment duration, and number of segments in a schedule. This application also collects detailed statistics and information about the experiments, and it stores them in structured XML log files. The collected data about each streaming session includes receiver hostname, current bandwidth estimate, timestamp, segment number, and segment length. This information is collected after all segments in a schedule have been received. This approach allows us to analyze all experimental data offline.

All nodes were controlled using the plDist PlanetLab script [24]. This script was used to install and configure java 1.6.0 as well as install, upgrade, start, and stop the prototype application. Since there were approximately 70 PlanetLab nodes used in the experiment, manual configuration would have been very inefficient. Using the plDist script made the setup easier.

Once the test parameters are specified, the test driver contacts the receiver's node administrative interface and the receiver node initiates a test run. Each test run spawns a new thread within the prototype program so that in theory multiple test runs could occur at the same time. During our experiments, however, we limited the test runs to one at a time. Since this prototype does not implement a complete P2P system, all senders are specified

Video Name	Size (MB)	Frame Count	Mean Bit Rate (kbps)	Description
South Park	26	30334	170	Low-quality, simple animation
Alladin	200	89998	440	High-quality, medium-complexity animation
Starship Troopers	270	89998	600	High-quality movie, low frame variability
Formula 1	190	44998	840	High-quality, medium frame variability sports video
Soccer	500	89998	1100	High-quality, high frame variability sports video

Table 5.1: Video streams used in the experiments.

explicitly for each scheduling test run. Since we are only evaluating the scheduling algorithm performance there is no need for the additional overhead and complexity. We also assume that no nodes are allowed to enter or leave the sender group during a test run. In a real-life system, where churn is expected, if a peer left the network during a streaming session, we would simply invoke the scheduling algorithm again to recompute the schedule using the updated peer membership information. In the course of a test run, the receiver first creates a schedule for the next segment window of the requested media file. The segment window size, specified by the test driver, is the number of segments considered for the current schedule where the duration of each segment is also specified by the test driver. During some preliminary experiments, we executed a relatively small set of test cases using window sizes of 10, 20, and 40 segments and found no major differences in the relative algorithms performance. Although the effects of using different window sizes should be examined in the future in more detail, our extensive experimental results focus only on the effects of varying video bit rates and number of senders, while keeping the segment window size at 20. Once the schedule is created, the receiver node creates a streaming session and each sender is delivered its respective schedule, i.e., the segments it is expected to transmit. Once a sender node receives a schedule, it creates a new thread that handles the segment transmission and the streaming session begins.

While the receiver node is receiving the media segments, the bandwidth estimates for each sender are continually updated. Once all the nodes in the requested schedule have been received or the segment window has almost expired (for 1 second segments and a 20 segment window size, this would occur after 20 seconds), a new schedule is created and the same process is repeated. Once a schedule session is finished, the results captured by the receiver are sent back to the test driver. The test driver then displays the results in text form as they become available and these results are also stored as XML log files at the test driver's location. The current implementation of the test driver does not provide a graphical

display of latest results. In all experiments, the same test was executed for each scheduling algorithm one after another so that the network conditions were as similar as possible for each algorithm's test run.

The test case results include meta data such as the media file, first and last segments in the current window, and scheduler type, in short all the meta data required to recreate the test case. The main part of the test case results, however, are the expected (as scheduled) and the actual (as received) segment traces. Once this data is stored at the receiver, it can be analyzed offline and we can look for various insights in the data when we graph and compare the different algorithms. We can see these graphed results and their comparison discussion in Section 5.4.

5.3 Performance Metrics

5.3.1 Continuity Index

Playback continuity is a primary objective for a streaming application and thus we use it as a basis of comparison between the Round-Robin, Random, Rarest First, and ODV scheduling algorithms. To evaluate continuity, we define the continuity index as the number of frames arrived before or on its playback deadline over the total number of frames; the exact formula is shown in Equation (5.1).

$$\text{Continuity Index} = \frac{f}{f_{total}}, \tag{5.1}$$

where f is the number of video frames received before or on the deadline, and f_{total} is total number of video frames.

From the definition, we can see that lost frames and late frames are treated the same way. Late or dropped frames cause glitches in the video playback and may cause the scene to be frozen. Since in all our experiments we used a reliable transport layer (TCP), there are no lost frames; only late frames.

Many previous works [70, 71] use some measure of the playback continuity such as our continuity index. The few papers that do mention the continuity index calculation, use segments instead of frames for these calculations. Using segments for calculating the continuity index leads to lower accuracy of the real playback continuity. In most cases, segments are

composed of many frames and it is quite possible for a segment to miss its deadline even though some of the frames contained within this segment have arrived on time. In essence, using segments for playback continuity calculations is not as accurate as using frames.

5.3.2 Balance Index

The load balancing metric, referred to as the balance index, attempts to describe the relative utilization of each sender peer in a streaming session. As opposed to the continuity index and the buffering delay, which are client side metrics, the balance index is more of a system metric indicating the relative loads assigned to different peers. It should also be noted that when we calculate the balance index, we use the segment counts as a basis for calculations. We propose the following Equation (5.2) to compute the balance index:

$$\text{Balance Index} = 1 - \sum_{i=1}^{P} \left| \frac{s_i}{s_{total}} - \frac{1}{P} \right| / bi_{max}^{P}, \tag{5.2}$$

where s_i is the number of segments received from peer i, s_{total} is the total number of segments received from all peers, P is the total number of peers, and bi_{max}^{P} is the maximum peer delivery ratio deviation for P senders.

Let us examine some of the terms in Equation (5.2). The term s_i/s_{total} is the ratio of segments delivered from peer i and $1/P$ is the ratio of segments delivered by any peer in a perfectly balanced scenario. The absolute value of the difference between these terms measures the deviation of the segments delivered by peer i from an ideally balanced delivery ratio. Let us now turn our attention to Eq. (5.3) that describes the term bi_{max}^{P}

$$bi_{max}^{P} = \sum_{j=1}^{P} \left| s_j - \frac{1}{P} \right| = 2 - \frac{2}{P}, \tag{5.3}$$

where $s_1 = 1$, $s_k = 0$ for $2 \leq k \leq P$, and $P \geq 2$.

This term is essentially the sum of the peer delivery ratio deviations from an ideally balanced delivery ratio in a scenario where all segments have been delivered by one peer. When we divide the sum of all peer deviations in Equation (5.2) by the term bi_{max}^{P}, we are essentially normalizing this sum's value. We subtract the result from Equation (5.2) to indicate that high balance index implies high load balancing and a low balance index implies low load balancing. In other words, a balance index of 1.0 indicates that all peers contributed equally to the streaming session, and a balance index of 0 indicates that all segments were

received from only one peer. Any value in between, indicates an intermediate level of sender utilization.

5.3.3 Buffering Delay

Buffering delay is an important characteristic of any P2P system. In our experiments, we compare the algorithms based on the amount of buffering delay that would have been required to achieve a continuity index of 1.0, i.e. the buffering delay required to receive all segments before their deadlines. Recall from Figure 3.1 and the related discussion that buffering delay moves the playout line up on the graph so that more frames are located below it. That implies that those frames are ready for playout as indicated by the playout line.

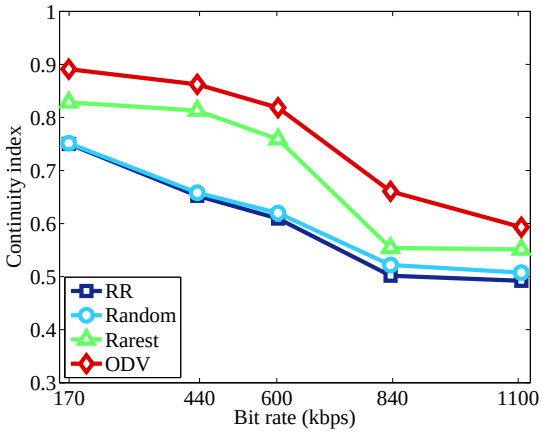
Most related research takes buffering delay into consideration when evaluating their P2P systems. Some scheduling algorithms [37] are actually designed with the goal of minimizing the buffering delay. No other works, however, consider buffering delay in addition to continuity and balance indices, i.e., they focus solely on the buffering delay metric.

5.4 Performance Results

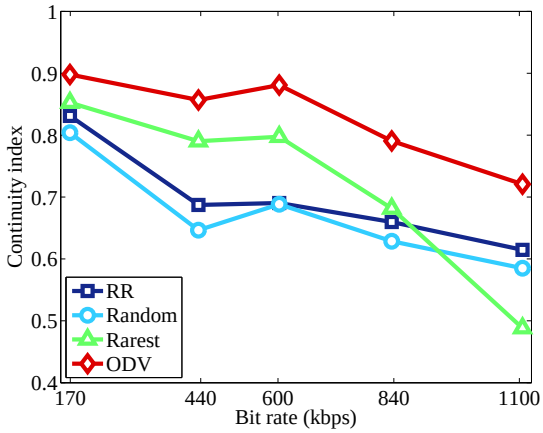
As described in Section 5.3, we consider three important performance metrics: continuity index, load balancing across peers, and buffering delay. The following sections present and analyze our experimental results for all four scheduling algorithms with regard to the above performance metrics.

5.4.1 Continuity Index Performance Results

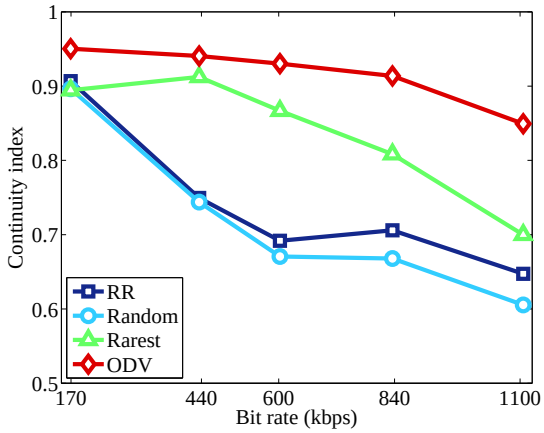
The continuity index captures an important angle of the user-perceived video quality, which is the smoothness of the rendered video streams. In Figure 5.2, we plot the continuity index computed for different scheduling algorithms using the considered five video streams, which have the average bit rates of 170, 440, 600, 840, and 1100 kbps. Different subfigures represent different number of senders in each streaming session. A few observations can be made on this figure. First, the Random and the Round-Robin algorithms have roughly the same low continuity index. This is because they do not consider the characteristics of peers and the network conditions in assigning segments to senders. The performance of these algorithms



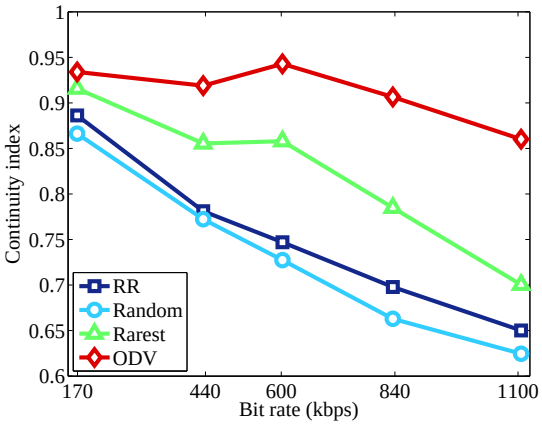
(a) Continuity index vs. streaming rate for 2 senders.



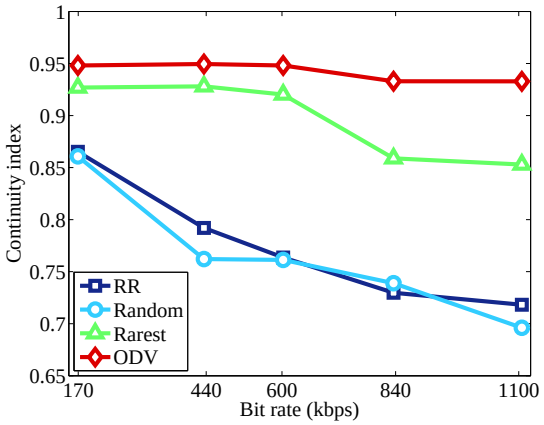
(b) Continuity index vs. streaming rate for 3 senders.



(c) Continuity index vs. streaming rate for 4 senders.



(d) Continuity index vs. streaming rate for 5 senders.



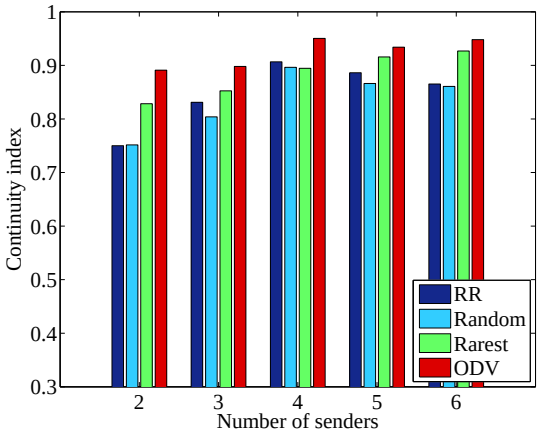
(e) Continuity index vs. streaming rate for 6 senders.

Figure 5.2: Continuity index vs. streaming rate for varying number of senders.

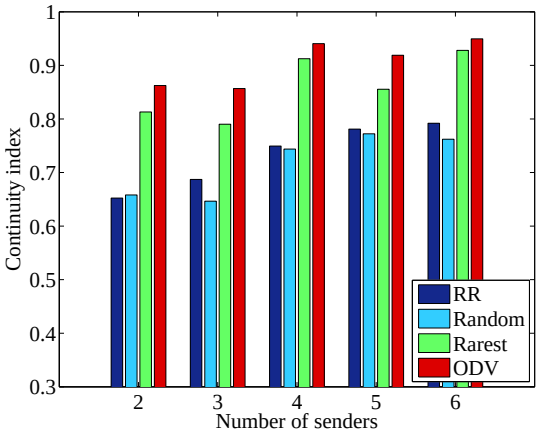
gets worse as the bit rate of the video increases. For example, in Figure 5.2(c), with four senders the continuity index is around 0.7 for videos with average bit rates of 840 kbps. This means that, on average, 30% of the frames miss their playback deadlines and the user-perceived quality will suffer significantly. As the bit rate increases, the importance of the scheduling algorithm becomes more apparent as there is more video data to be transmitted.

The second observation on Figure 5.2 is that our proposed ODV algorithm consistently outperforms the Rarest First algorithm. For example, in Figure 5.2(c), the ODV algorithm achieves a continuity index of about 0.85 for high-quality video streams (with bit rate of 1100 kbps) with four senders, while the Rarest First algorithm achieves only a continuity index of 0.7. In addition, as shown by all subfigures in Figure 5.2, the gap between our ODV algorithm and other algorithms increases as the bit rate of the video increases, which is expected in future P2P streaming systems as users continually demand better quality and higher resolution videos. Therefore, our proposed algorithm will yield even better performance for future P2P streaming systems.

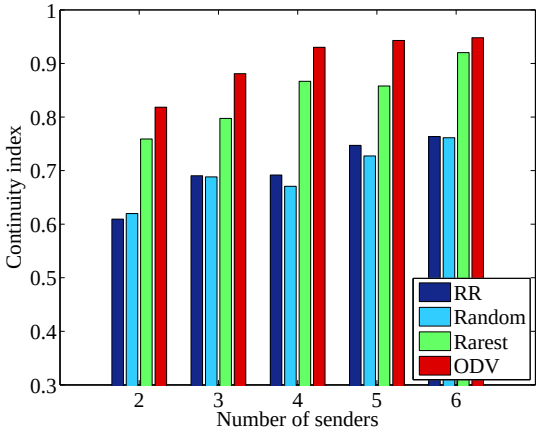
As another observation, we should note that the streaming bit rate is inversely proportional to the continuity index. This effect is independent of the number of senders and the scheduling algorithm used. This makes perfect sense since as we increase the streaming bit rate, we have more data to transfer in the same amount of time and this obviously gets progressively more difficult. We should also observe that different scheduling algorithms respond differently when the system is under progressively increased stress. We can see that



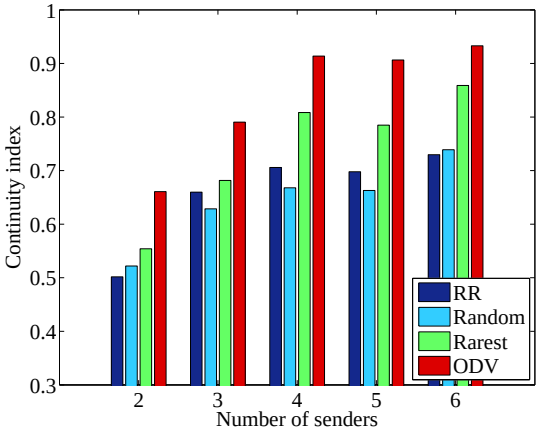
(a) Continuity index vs. number of senders for streaming of the South Park movie (170 kbps).



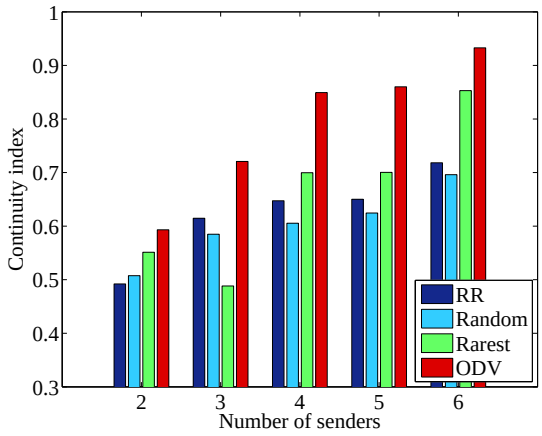
(b) Continuity index vs. number of senders for streaming of the Alladin movie (440 kbps).



(c) Continuity index vs. number of senders for streaming of the Starship Troopers movie (600 kbps).



(d) Continuity index vs. number of senders for streaming of Formula 1 video (840 kbps).



(e) Continuity index vs. number of senders for streaming of a soccer video (1100 kbps).

Figure 5.3: Continuity index vs. number of senders for Starship Troopers, South Park, and Alladin movies streaming sessions.

at lower streaming rates all algorithms produce similar results whereas at the higher streaming rates, the choice of scheduling algorithm starts to make a difference and the Rarest First and ODV algorithms outperform the Random and Round-Robin algorithms by a greater margin.

As a final observation on Figures 5.2 and 5.3, increasing the number of senders generally improves the continuity index, which is intuitive as more senders bring in more streaming capacity. These effects of the increasing number of senders are most evident at higher bit rates. We can see this clearly by comparing Figures 5.3(a) and 5.3(e). In Figure 5.3(a) the difference between the continuity index values for a 2-sender session and a 6-sender session is about 10-15%, for any algorithm. Also, at this lowest streaming bit rate the relative performance results for all scheduling algorithms are similar, i.e. the characteristics of the Rarest First and ODV algorithms do not become fully evident. In Figure 5.3(e), however, the difference between the 2-sender session and the 6-sender session is approximately 30-35%.

5.4.2 Balance Index Performance Results

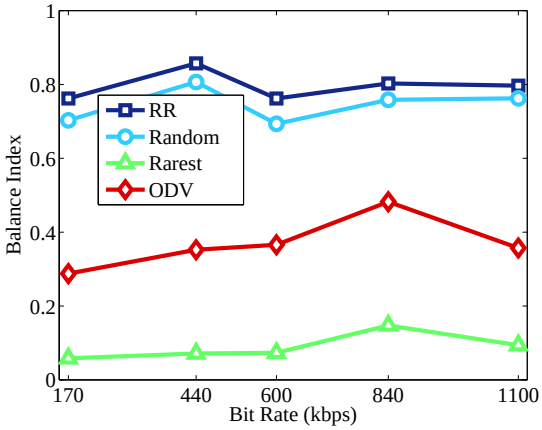
The results for the load balancing index of all algorithms is demonstrated in Figures 5.4 and 5.5. As expected, these figures show that the Round-Robin and the Random algorithms achieve a high load balancing index, which is almost 1.0. This is because these algorithms

make all senders contribute equally without regard to the quality of the streaming session or the capacity of the senders. While the load balancing is a desirable property, it should not be used to sacrifice the ultimate goal of achieving good streaming quality. That is, a reasonable, not too skewed load balancing index is acceptable as far as the quality is not compromised. Figures 5.4 and 5.5 show that our proposed ODV algorithm significantly improves the load balancing index beyond that of the Rarest First algorithm. The Rarest First algorithm stresses the fast peers too much by saturating them with segment requests first before allocating any requests to slower peers. This results in a very skewed load distribution on peers, which may discourage peers from contributing resources. Our algorithm, on the other hand, does not ignore slower peers and assign to them the segments that they can deliver on time. This in turn reduces the load on faster peers and results in more balanced load distribution across all peers. Finally, from looking at the load balancing index results in all subfigures, we can infer that this metric is quite independent of the streaming bit rate and the number of senders. The load balancing index is an intrinsic property of the scheduling algorithm and it is not affected by the experiment variables.

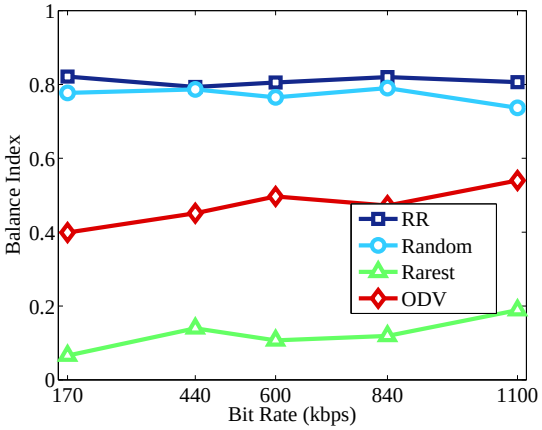
5.4.3 Buffering Delay Performance Results

Figures 5.6 and 5.7 show the effects of the scheduling algorithms on the buffering delay required to achieve smooth playback of the videos, again with different number of senders and for various video streams. These figures show that the Round-Robin and the Random algorithms require substantial buffering delays. On the other hand, the proposed ODV and Rarest First algorithms require much smaller buffering delays, less than 10 seconds in all cases. The buffering delay is decreased to below 3 seconds by increasing the number of senders to six as shown in Fig. 5.6(e). The results in Figures 5.6 and 5.7 also show that our ODV algorithm always produces smaller or the same buffering delay as the Rarest First algorithm.

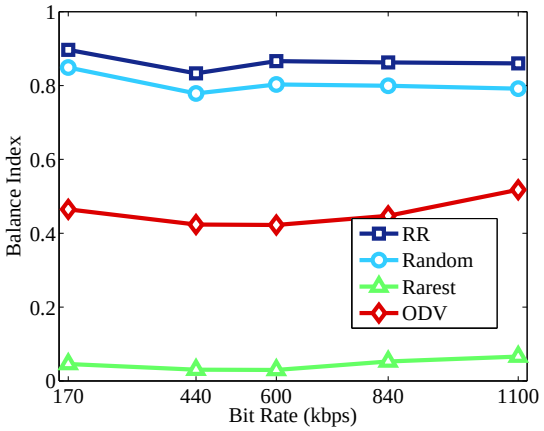
When examining the results for the Round-Robin and Random algorithms, we can clearly see that increasing the streaming rate leads to greater buffering delays. This is consistent with previous results and again, stems from the fact that at higher bit rates we are transferring more data, which requires more time. The trends for both these algorithms are exactly the same, i.e., independently of the number of senders, the buffering delay increases with an increasing streaming bit rate. In all cases, the Round-Robin algorithm outperforms the Random algorithm but we should also note another interesting trend related to the relative



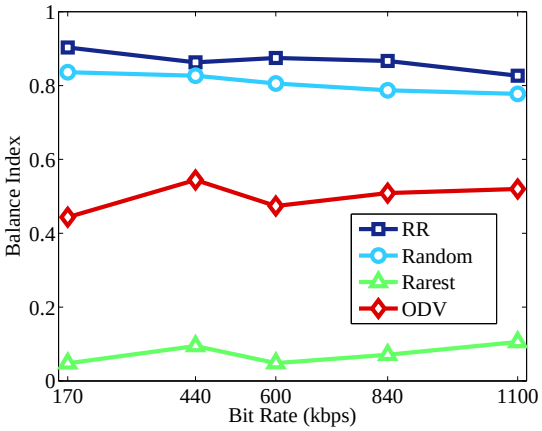
(a) Balance index vs. streaming bit rate for 2-sender streaming session.



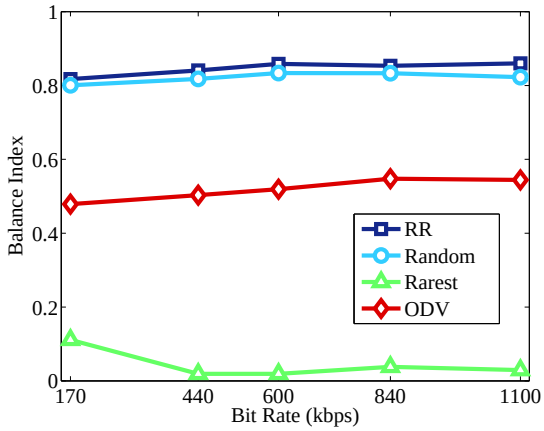
(b) Balance index vs. streaming bit rate for 3-sender streaming session.



(c) Balance index vs. streaming bit rate for 4-sender streaming session.



(d) Balance index vs. streaming bit rate for 5-sender streaming session.



(e) Balance index vs. streaming bit rate for 6-sender streaming session.

Figure 5.4: Balance index vs. bit rate for variable streaming rate sessions.

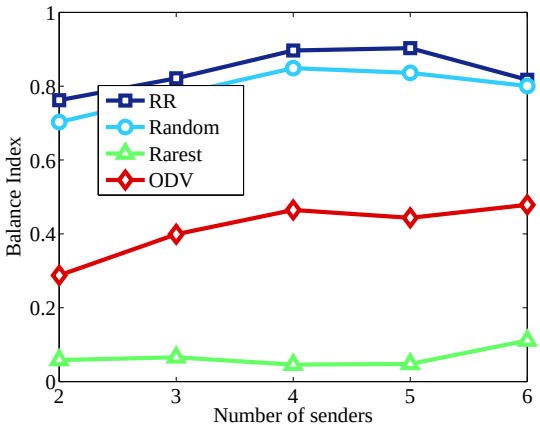
performance of these two algorithms. In streaming sessions with less senders, the performances of these two algorithms are really close. In streaming sessions with more senders, there is a larger difference between the buffering delay required when using the Random algorithm and when using the Round-Robin algorithm.

Let us now examine the results for the Rarest First and the ODV algorithms shown in Figure 5.6. First off, we clearly see that the buffering delay required for these two algorithms is significantly smaller than the values required for the Round-Robin and Random algorithms. We also see that, although not as clearly defined and prominent, the buffering delay also increases with higher streaming bit rates for these two algorithms. We can see that the ODV algorithm generally results in smaller buffering delays than the Rarest First algorithm but this difference is not very prominent in most cases. Although, as mentioned in Section 5.4, our results are based on large number of aggregate data,

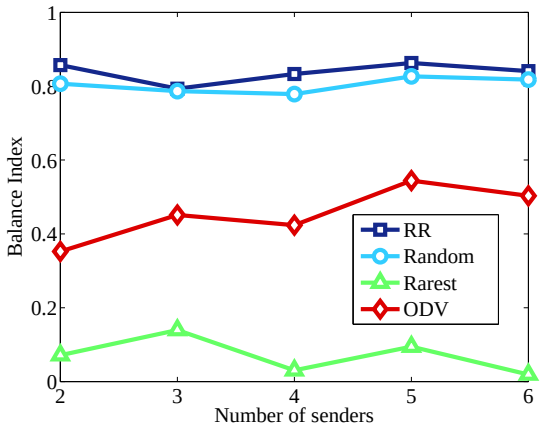
there are more anomalies

we can see that there are more anomalies in the case of buffering delay than in the cases of the balance or continuity indices. This could be the result of the buffering delay being a more sensitive characteristic of the system to network fluctuations. To alleviate this effect, we will need to perform more experiments in the future to collect larger quantities of data so that the average will show smoother trends.

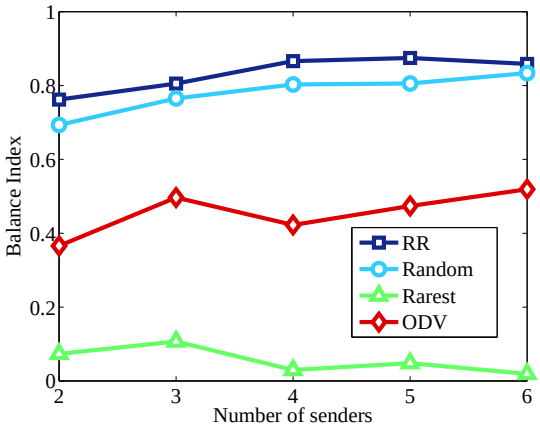
Let us now turn our attention to Figure 5.7 and look in more detail at the effects of



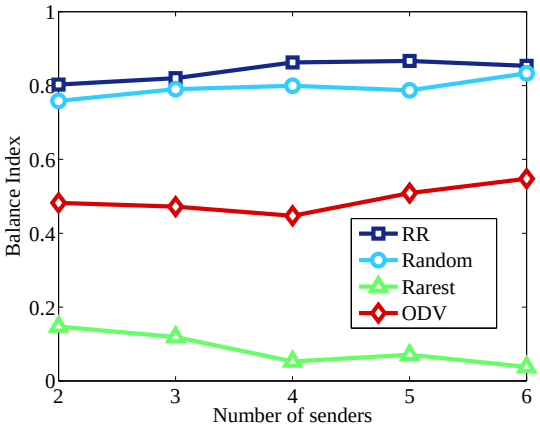
(a) Balance index vs. bit rate for streaming of the Southpark movie (170 kbps).



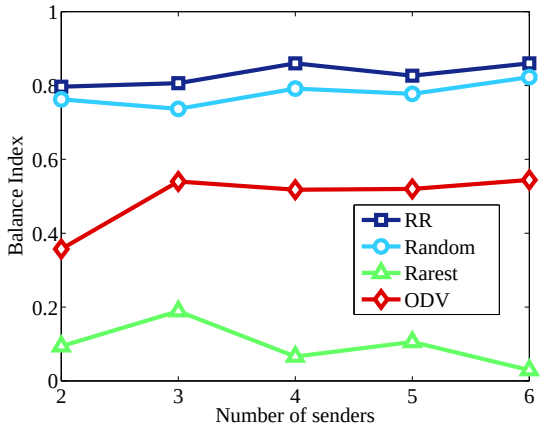
(b) Balance index vs. bit rate for streaming of the Alladin movie (440 kbps).



(c) Balance index vs. bit rate for streaming of the Starship Troopers movie (600 kbps).



(d) Balance index vs. bit rate for streaming of Formula 1 racing video (840 kbps).

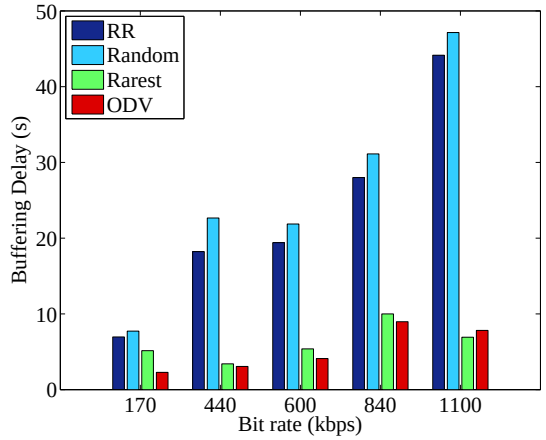


(e) Balance index vs. bit rate for streaming of Formula 1 racing video (1100 kbps).

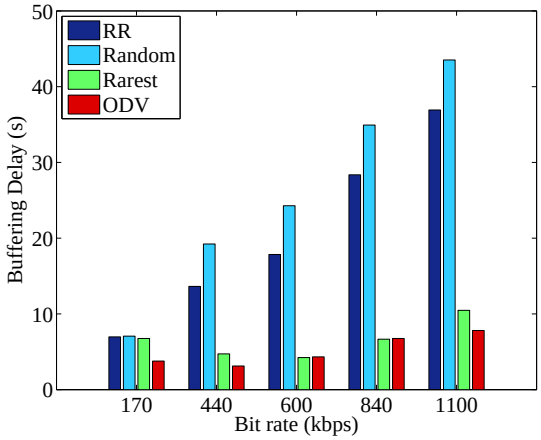
Figure 5.5: Balance index vs. bit rate for streaming sessions with variable number of senders.

different number of senders in a streaming session on the amount of buffering delay required. When we examine the results for the Random algorithm, we realize that it is hard to notice any sort of a regular trend. It seems that number of senders in the streaming session does not have a significant effect on the buffering delay when using the Random algorithm. This is somewhat similar in the case of the Round-Robin algorithm but we do start to notice some smaller trends in this case. This is more evident at higher streaming bit rates where the buffering delay starts decreasing with an increasing number of senders. This trend is more visible in Figures 5.7(d) and 5.7(e) than in Figures 5.7(a) and 5.7(b). Again, the Round-Robin algorithm outperforms the Random algorithm.

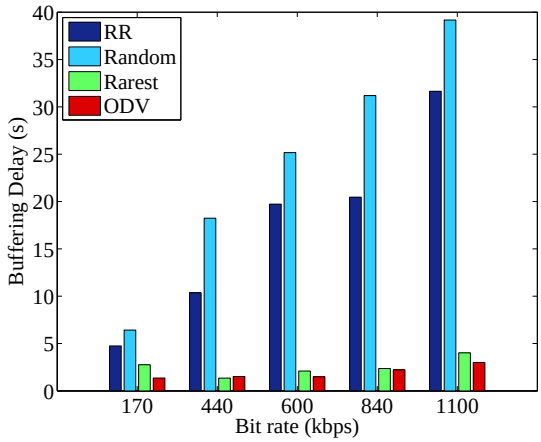
As for the Rarest First and the ODV algorithms, in general, we observe that the ODV algorithm requires smaller buffering delays, although this difference is not very significant in most cases. The trends for these two algorithm are somewhat similar to those of Round-Robin and Random algorithms with regard to the effect of increasing the number of senders at lower bit rates as compared to the effects at higher bit rates; more senders definitely decrease the buffering delay. This stems from the fact that both these algorithm use bandwidth estimates and more efficient segment assignments to actually take advantage of the extra senders available. Again, even at higher streaming rates, there are a couple of anomalies in the data indicating that the buffering delay is more sensitive to external factors such as network fluctuations than the continuity and the balance indices.



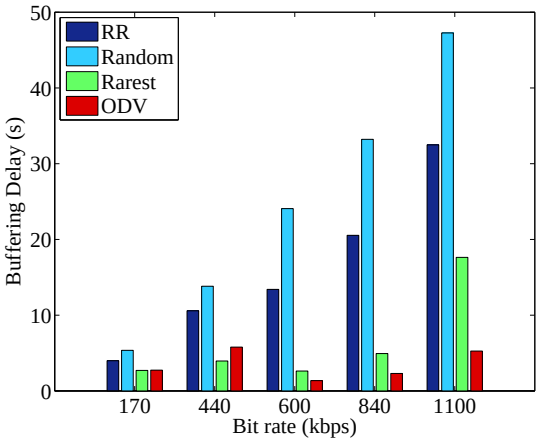
(a) Buffering delay vs. streaming bit rate for 2-sender streaming session.



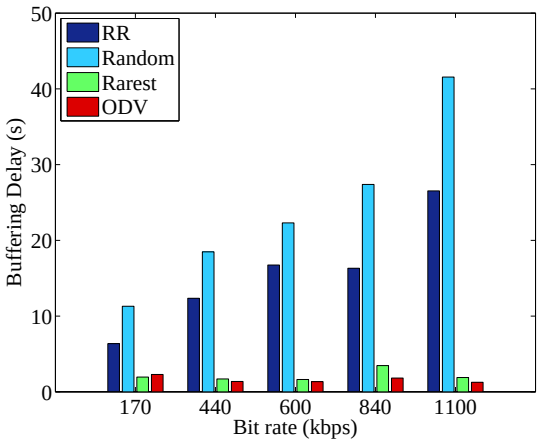
(b) Buffering delay vs. streaming bit rate for 3-sender streaming session.



(c) Buffering delay vs. streaming bit rate for 4-sender streaming session.

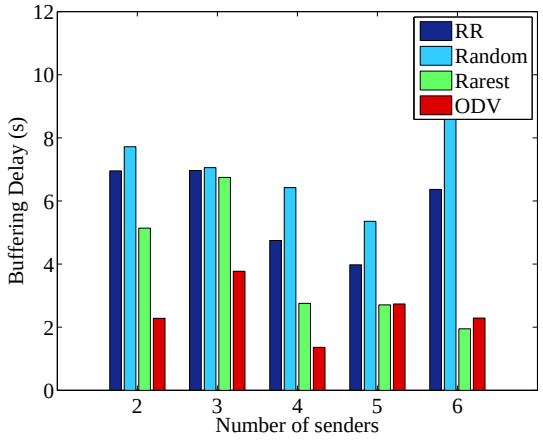


(d) Buffering delay vs. streaming bit rate for 5-sender streaming session.

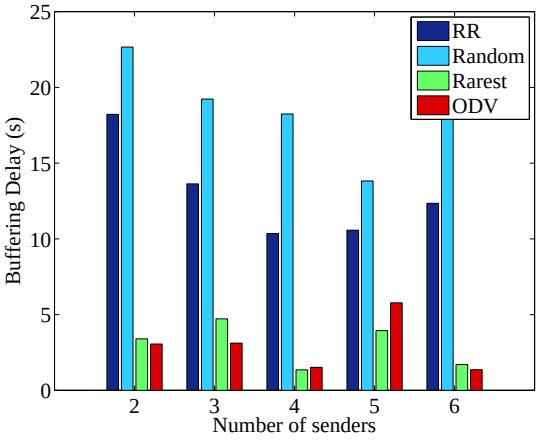


(e) Buffering delay vs. streaming bit rate for 6-sender streaming session.

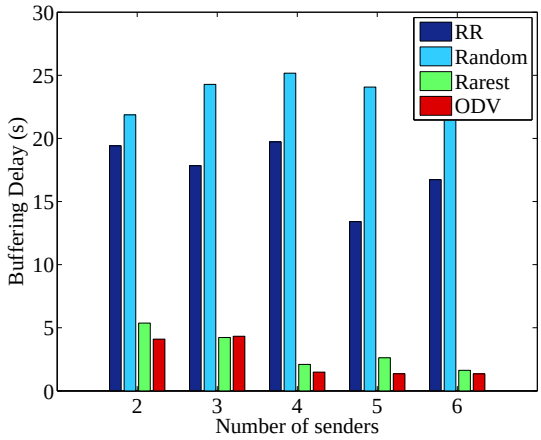
Figure 5.6: Buffering delay required for continuity index of 1.0 vs. bit rate for variable streaming rate sessions.



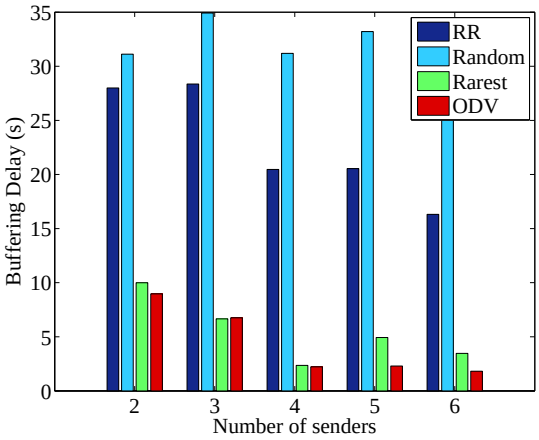
(a) Buffering delay vs. bit rate for streaming of the Southpark movie (170 kbps).



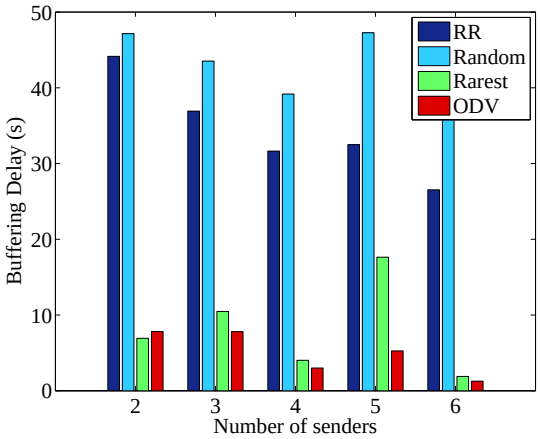
(b) Buffering delay vs. bit rate for streaming of the Alladin movie (440 kbps).



(c) Buffering delay vs. bit rate for streaming of the Starship Troopers movie (600 kbps).



(d) Buffering delay vs. bit rate for streaming of Formula 1 racing video (840 kbps).



(e) Buffering delay vs. bit rate for streaming of soccer video (1100 kbps).

Figure 5.7: Buffering delay required for continuity index of 1.0 vs. bit rate for streaming sessions with variable number of senders.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

We have experimentally analyzed three existing segment transmission scheduling algorithms in P2P streaming systems: Round-Robin, Random, and Rarest First. Our analysis was done by implementing a prototype P2P streaming system and deploying it on more than 70 PlanetLab nodes distributed all over the world. We conducted over 500 experiments with different video streams and number of sender peers. We measured media streaming performance metrics such as continuity index, load balance index, and buffering delay for all algorithms. Our analysis confirms that the segment scheduling algorithms have a significant impact on the user-perceived visual quality in P2P streaming systems. Several lessons were drawn from our analysis. For example, we showed that the Rarest First algorithm may overload fast sender peers while leaving the slower ones underutilized, which may lead to small continuity index and could also discourage peers from contributing resources to the P2P streaming system.

In addition, we proposed a new segment transmission scheduling algorithm, which we call On-time Delivery of VBR streams (ODV). ODV considers the variable nature of video streams, and tries to maximize the number of segments that meet their deadlines by assigning them to peers that will deliver them earlier. Our extensive PlanetLab experiments showed that ODV results in a continuity index that is 35—40% higher than Round-Robin and Random algorithms, and 20—25% higher than the Rarest First algorithm. Our results also show that the buffering delay for ODV is only up to 8% of the time required for the Round-Robin and Random algorithms, and, on average, is about only 40% of the time

required by the Rarest First algorithm. Furthermore, the ODV algorithm achieves much better load balancing index than the Rarest First algorithm: up to 5 times improvement is observed in our experiments. Therefore, our proposed ODV algorithm not only provides better video quality, but also yields more efficient utilization of the peers' resources, which in turn enhances the stability and scalability of a P2P system.

We mention above that our proposed scheduling algorithm, ODV, uses VBR video meta data during the scheduling process. As described in Chapter 4, under the ODV algorithm the sender for every segment is selected partially based on the current estimated load of each sender. The estimated load of a sender is the amount of work that the sender has already been committed to by the schedule and this amount of work depends on the segment size and the sender's bandwidth. In order to be able to accurately calculate the length of each segment, the scheduler is provided with the VBR video meta data that describes the frame and segment sizes for a particular portion of the video. This aspect of the ODV algorithm can be easily transferred to other scheduling algorithms such as Rarest First. Our method for VBR support during segment scheduling can thus easily enhance other scheduling algorithms and existing P2P streaming system.

Aside from these direct contributions, we have also devised a scheduling algorithm comparison framework that uses many more performance metrics than existing work. This algorithm comparison framework allows us to analyze in detail the strengths and weaknesses of each scheduling algorithm, both quantitatively and qualitatively. Our quantitative analysis, shown in Chapter 5, provides graphical results of all our experiments, and is the source of the performance differences mentioned above.

Our graphical tools for qualitative scheduling algorithm comparison, called streaming traces or schedule trace graphs, allow detailed analysis of every streaming session. When comparing various scheduling algorithms, it is important to gain deeper insight into the details of their operation at the lower level so that we can understand and predict their resulting behavior at a more macroscopic scale. In our case, it is extremely useful to be able to examine the transmission of every video frame from the senders to the receiver. We use the schedule trace graphs, described in detail in Section 3.2, to demonstrate and explain the real-life effects of the algorithms on the video frame delivery.

Finally, we also created an algorithm testing framework that improves the automation and efficiency of new scheduling algorithm development. This framework includes a test driving application, which is an administrative interface that allows researchers to manage

a large number of experiments from a central location. This test driver also enabled us to create and execute various test cases while allowing control over test parameters such as the receiver node, sender nodes, video stream, scheduling algorithm, segment duration, and number of segments in a schedule. Our framework also handles the collection and storage of experiment data that allows us to create graphical representations and consequently analyze the results offline.

6.2 Future Work

The work presented in this thesis shows many promising results. There are, however, many areas of this research that can be enhanced in the future. The current prototype P2P system was reduced to a scheduling testing framework. In future research, however, it would be interesting to perform the algorithm comparison on a complete P2P system that includes components such as constructing the overlay network, node membership, and sender selection. In a complete system, we could analyze the interaction of the scheduling algorithm with the other components of the system. In our current experiments we used 70 different PlanetLab nodes but we could use more nodes in order to examine the macro effects of the local segment scheduling decisions.

Another possible area of algorithm comparison improvement is to examine the algorithm behavior under more stressful conditions. For example, instead of using a single receiver within a streaming session, we could let all nodes act as senders and receivers simultaneously. Although this capability was built into our P2P prototype application, we did not actually make use of this functionality in our test cases. We also saw that the strength of the algorithm was demonstrated the most at higher streaming bit rates. In our current experiments, the highest streaming bit rate used was approximately 1100 kbps but for the sake of algorithm comparison, we could use even higher bit rates.

Many other scheduling algorithms have been devised for P2P streaming systems and we only used three existing algorithms as the basis for our comparison. There is no reason not to implement many of the other existing scheduling algorithms to see how their performance compares against the others in a system where, aside from the experiment parameters, the scheduling algorithm is the only variable. Along the same lines, there is no reason not to develop and experiment with more, new scheduling algorithms. Since evaluating a scheduling algorithm has become mostly an automated task within our testing framework

and prototype system, it would not be difficult to try other scheduling approaches.

Our current experiment variables include the scheduling algorithm, number of senders, and streaming bit rate. We graphically demonstrated their effects on the performance metrics in Chapter 5. There were, however, two other experiment parameters that stayed constant in all experiments; those parameters were segment window size and segment duration/size. Segment window size is the size of the scheduling window in terms of the number of video segments during a single algorithm execution. The window size used for all experiments presented in this work is 20. However, it would be interesting to see whether the window size itself has any significant effects on the performance metrics and behavior of the system in general. One obvious effect would be that with increasing segment window size, the scheduling would occur less frequently and thus the peer bandwidth estimates would be more likely to decrease in accuracy. This could in turn lead to inefficient schedules and ultimately affect the performance metrics and user-perceived video quality.

Segment duration is the length of time included in one video segment used by the scheduler. In our experiments, we used segment duration of one second. This constant was arbitrary and once again, it would be interesting to determine its potential effects on the rest of the system, including the performance metrics. Another interesting aspect regarding the segment duration is that it is not necessary to keep it constant through the system or even within a streaming session. The scheduling algorithm could very well use fine-grained control over the segment duration in order to optimize the scheduling process. Using a constant segment duration is definitely a disadvantage because it prevents the scheduling algorithm to reassign resources to different senders at a fine scale.

During the analysis of existing algorithms and the proposed algorithm ODV, in Chapters 3 and 4, we mentioned that only ODV uses the VBR video meta data during the scheduling process. It would be interesting to determine whether using the VBR video meta data has any significant effect on the system and the resulting performance metrics when it is used a variable for testing of each algorithm.

Our testing framework could include several useful enhancements such as graphical display of near real-time scheduling results. Although, this feature would require extra test driver application development, the near instantaneous feedback could potentially enhance the algorithm design process since the researcher would not need to wait for feedback.

Finally, one weakness in our current prototype system is that any changes to the scheduling algorithms require redeployment of new application binaries. With the help of plDist

scripts this process is mostly automated, it is caused by the lack of maturity of the prototype application. We should be able to enhance the prototype system to potentially accept the scheduling algorithm binary files themselves as part of the test case execution. Such dynamic, run-time behavior would provide our testing framework with greater flexibility and simplified deployment of new functionality.

Bibliography

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

[2] 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.

[3] A. Asiki, K. Doka, I. Konstantinou, A. Zissimos, and N. Koziris. A distributed architecture for multi-dimensional indexing and data retrieval in grid environments. In *Proc. of Cracow 07 Grid Workshop*, Cracow, Poland, October 2007.

[4] S. Banerjee, B. Bhattacharjee, C. Kommareddy, and G. Varghese. Scalable application layer multicast. In *Proc. of Annual Conference of the ACM Special Interest Group on Data Communication (SIGCOMM'02)*, pages 205 – 220, Pittsburgh, PA, USA, August 2002.

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

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

[7] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: Highbandwidth multicast in cooperative environments. In *Proc. of 19th ACM Symposium on Operating Systems Principles (SOSP'03)*, Bolton Landing, NY, USA, October 2003.

[8] 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)*, San Jose, CA, USA, January 2009.

[9] J. Chan, V. Li, and K. Lui. Scheduling algorithms for peer-to-peer collaborative file distribution. In *Proc. of International Conference on Collaborative Computing: Networking, Applications and Worksharing*, pages 1 – 10, San Jose, CA, USA, December 2005.

[10] J. Chen, J. Wu, C. Shih, and T. Kuo. Approximation algorithms for scheduling multiple feasible interval jobs. In *Proc. of IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 11 – 16, Hong Kong, China, August 2005.

[11] Z. Chen, K. Xue, and P. Hong. A study on reducing chunk scheduling delay for mesh-based p2p live streaming. In *Proc. of 7th International Conference on Grid and Cooperative Computing*, pages 356 – 361, Shenzhen, China, October 2008.

[12] H. Chi, Q. Zhang, J. Jia, and X. Shen. Efficient search and scheduling in P2P-based media-on-demand streaming service. *IEEE Journal On Selected Areas in Communications*, 25(1):119 – 130, January 2007.

[13] Y.-H. Chu, S. Rao, S Seshan, and H. Zhang. A case for end system multicast. In *Proc. of International Conference on Measurement and Modeling of Computer Systems*, Santa Clara, CA, USA, June 2000.

[14] The Coral Content Distribution Network Home Page. `http://www.coralcdn.org`.

[15] CPUShare Home Page. `http://www.cpushare.com`.

[16] J. Crowcroft and I. Pratt. Peer to peer: Peering into the future, May 2002.

[17] F. Dabek, M. Kaashoek, D. Karger, D. Morris, and I. Stoica. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of ACM SIGCOMM'01*, San Diego, CA, USA, August 2001.

[18] F. Dabek, M. Kaashoek, D. Karger, D. Morris, and I. Stoica. Wide-area cooperative storage with cfs. In *Proc. of 18th ACM Symposium on Operating Systems Principles (SOSP'01)*, pages 202 – 215, Banff, Canada, October 2001.

[19] eMule Home Page. `http://www.emule-project.net`.

[20] J. Kubiatowicz et al. Oceanstore: An architecture for global-scale persistent storage. In *Proc. of 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'00)*, pages 190 – 201, Boston, MA, USA, November 2000.

[21] Freenet Home Page. `http://freenetproject.org`.

[22] M. R. Garey and D. J. Johnson. *Computers and intractability: a guide to the theory of NP-completeness*. W.H. Freeman, New York, USA, 1979.

[23] Genome@Home Home Page. `http://www.stanford.edu/group/pandegroup/genome`.

[24] M. Georg. pldist: tool for parallel distribution of files to planetlab nodes, May 2005. `http://www.arl.wustl.edu/~mgeorg/plDist.html`.

[25] Gnutella Wikipedia Page. `http://en.wikipedia.org/wiki/Gnutella`.

[26] K. Gummadi, R. Dunn, S. Saroiu, S. Gribble, H. Levy, and J. Zahorjan. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In *Proc. of ACM Symposium on Operating Systems Principles (SOSP'03)*, Bolton Landing, NY, USA, October 2003.

[27] M. Hefeeda, A. Habib, D. Xu, B. Bhargava, and B. Botev. Collectcast: A peer-to-peer service for media streaming. *Multimedia Systems*, 11(1):68 – 81, November 2005.

[28] 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.

[29] L. Hwang, M. Cai, Y.-K. Kwok, S. Song, Y. Chen, and Y. Chen. Dht-based security infrastructure for trusted internet and grid computing. *International Journal of Critical Infrastructures*, 2(4):412 – 433, November 2006.

[30] iMesh Home Page. `http://www.imesh.com`.

[31] J. Jannotti, D. Gifford, K. Johnson, M. Kaashoek, and J. O'Toole. Overcast: Reliable multicasting with an overlay network. In *Proc. of Operating Systems Design and Implementation (OSDI'00)*, pages 197 – 212, San Diego, CA, USA, October 2000.

[32] Kademlia: A Design Specification. `http://xlattice.sourceforge.net/components/protocol/kademlia/specs.html`.

[33] M. Kamath, K. Ramamritham, and D. Towsley. Continuous media sharing in multimedia database systems. In *Proc. of 4th International Conference on Database Systems for Advanced Applications (DASFAA'95)*, Singapore, April 1995.

[34] T. Karagiannis, A. Broido, N. Brownlee, K. C. Claffy, and M. Faloutsos. Is P2P dying or just hiding? In *Proc. of IEEE Global Telecommunications Conference (GLOBECOM'04)*, pages 1532 – 1538, Dallas, TX, USA, November 2004.

[35] Kazaa Home Page. `http://www.kazaa.com`.

[36] 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)*, San Diego, CA, USA, December 2009.

[37] J. Kwon and H. Yeom. Distributed multimedia streaming over peer-to-peer networks. In *Euro-Par 2003 Parallel Processing*, pages 851 – 858, Klagenfurt, Austria, August 2003.

[38] 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.

[39] LimeWire Home Page. `http://www.limewire.com`.

[40] 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.

[41] 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.

[42] N. Magharei, Y. Guo, and R. Rejaie. Issues in offering live p2p streaming service to residential users. In *Proc. of Consumer Communications and Networking Conference (CCNC'07)*, pages 757 – 762, Las Vegas, NV, USA, January 2007.

[43] N. Magharei, A. Rasti, and D. Stutzbach. Prime: Peer-to-peer receiver-driven mesh-based streaming. In *Proc. of IEEE International Conference on Computer Communications (INFOCOM'07)*, pages 1415 – 1423, Anchorage, Alaska, USA, May 2007. INFOCOM 2007.

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

[45] D. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, and B. Richard. Peer-to-peer computing, March 2002.

[46] Napster Web Page. `http://en.wikipedia.org/wiki/Napster`.

[47] 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.

[48] T. Piotrowski, S. Banerjee, S. Bhatnagar, S. Ganguly, and R. Izmailov. Peer-to-peer streaming of stored media: The indirect approach. In *Proc. of the joint International Conference on Measurement and Modeling of Computer Systems*, pages 371 – 372, Saint-Malo, France, June 2006.

[49] PlanetLab Web Page. `http://www.planet-lab.org`.

[50] PPLive Web Page. `http://www.arl.wustl.edu/~mgeorg/plDist.html`.

[51] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, and J. Kubiatowicz. Pond: The oceanstore prototype. In *Proc. of 2nd USENIX Conference on File and Storage Technologies (FAST'03)*, San Francisco, CA, USA, March 2003.

[52] 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.

[53] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. of 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware'01)*, Heidelberg, Germany, November 2001.

[54] A. Rowstron and P. Druschel. Storage management in past, a large-scale, persistent peer-to-peer storage utility. In *Proc. of 18th ACM Symposium on Operating Systems Principles (SOSP'01)*, Banff, Canada, October 2001.

[55] S. Saroiu, K. Gummadi, R. Dunn, S. Gribble, and H. Levy. An analysis of internet content delivery systems. In *Proc. of 5th Symposium on Operating Systems Design and Implementation (OSDI'02)*, Boston, MA, USA, December 2002.

[56] R. Schollmeier. A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In *Proc. of the First International Conference on Peer-to-Peer Computing (P2P'01)*, Lingkoping, Sweden, August 2001.

[57] SETI@Home Home Page. `http://setiathome.berkeley.edu`.

[58] E. Setton, J. Noh, and B. Girod. Low latency video streaming over peer-to-peer networks. In *Proc. of International Conference on Multimedia and Expo (ICME'06)*, pages 757 – 762, Toronto, Canada, July 2006.

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

[60] D. Tran, K. Hua, and T. Do. Zigzag: An efficient peer-to-peer scheme for media streaming. In *Proc. of IEEE INFOCOM*, San Francisco, CA, USA, April 2003.

[61] 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.

[62] uTorrent Home Page. `http://www.utorrent.com`.

[63] Video Traces Web Page. `http://www.tkn.tu-berlin.de/research/trace/trace.html`.

[64] Peer-to-peer Wikipedia Page. `http://en.wikipedia.org/wiki/Peer-to-peer`.

[65] D. Xu, M. Hefeeda, S. Hambrusch, and B. Bhargava. On peer-to-peer media streaming. *Purdue Computer Science Technical Report*, April 2002.

[66] 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.

[67] YaCy Home Page. `http://yacy.net`.

[68] B. Yang and H. Garcia-Molina. Designing a super-peer network. In *Proc. of 9th International Conference on Data Engineering (ICDE'09)*, Bangalore, India, March 2008.

[69] J. Yin, W. Yao, L. Ma, and J. Dong. CoopStreaming: a novel peer-to-peer system for fast live media streaming. In *Proc. of International Conference on Advances in Web-Age Information Management (WAIM'05)*, pages 882 – 887, Hangzhou, China, October 2005.

[70] 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.

[71] 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 International Conference on Computer Communications (INFOCOM'05)*, pages 2102 – 2111, Miami, FL, March 2005.

[72] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, and J. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41 – 53, January 2004.