Efficient Algorithms for Multi-Sender Data Transmission in Swarm-Based Peer-to-Peer Streaming Systems

Yuanbin Shen, Cheng-Hsin Hsu, *Member, IEEE*, and Mohamed Hefeeda, *Senior Member, IEEE*

***Abstract*—In mesh-based peer-to-peer (P2P) streaming systems, each video sequence is divided into segments, which are then streamed from multiple senders to a receiver. The receiver needs to coordinate the senders by specifying a transmission schedule for each of them. We consider the problem of scheduling segment transmission in P2P streaming systems, where different segments have different weights in terms of quality improvements to the received video. Our goal is to compute the transmission schedule for each receiver in order to maximize the perceived video quality. We first show that this scheduling problem is NP-Complete. We then present an integer linear programming (ILP) formulation for it, so that it can be solved with any ILP solver. This optimal solution, however, is computationally expensive and is not suitable for real-time P2P streaming systems. Thus, we propose two approximation algorithms to solve this segment scheduling problem. These algorithms provide theoretical guarantees on the worst-case performance. The first algorithm considers the weight of each video segment. The second algorithm is simpler and it assumes that segments carry equal weights. We analyze the performance and complexity of the two algorithms. In addition, we rigorously evaluate the proposed algorithms with simulations and experiments using a prototype implementation. Our simulation and experimental results show that the proposed algorithms outperform other algorithms that are commonly used in deployed P2P streaming systems and that have been recently proposed in the literature.**

***Index Terms*—Peer-to-peer networks, quality optimization, transmission scheduling.**

I. Introduction

PEER-TO-PEER (P2P) streaming systems [1]–[4] have been proposed to distribute multimedia contents at low infrastructure costs. P2P streaming systems can be built in two different ways [5]: 1) tree-based systems in which one or more trees are constructed to connect peers for transferring contents [6]–[8] and 2) mesh-based systems in which each peer connects to a few neighboring peers without an explicit network topology before exchanging data with each other [1], [9], [10]. Recent studies show that mesh-based systems incur lower maintenance overhead, adapt better to network dynamics, are easier to implement [11], and lead to better perceived video quality [12]. More importantly, mesh-based systems are widely deployed, and thus, we study mesh-based systems in this work.

In mesh-based systems, a video sequence is partitioned into small *segments*, and segments are transmitted from multiple senders to a receiver. The receiver must coordinate the segment transmission from its senders. More precisely, a receiver runs a *scheduling* algorithm to compose a transmission schedule for its senders, which specifies for each sender the assigned segments and their transmission times. Previous works [13], [14] show that when the resources (especially bandwidth) are enough to stream the videos in a P2P streaming system, almost all existing scheduling algorithms perform equally well. With the introduction of more and higher quality video streams over the Internet, resources are never sufficient; thus, more intelligent scheduling algorithms are needed to fully utilize the limited resources in P2P streaming systems.

Composing segment transmission schedules is not an easy task, as P2P streaming systems impose time constraints on segment transmission. Segments arriving at the receiver after their decoding deadlines are not useful, because they cannot be rendered to users for improving video quality. Hence, segment scheduling algorithms should strive to maximize the perceived video quality delivered by the on-time segments. Optimally constructing segment schedules is computationally expensive [9], and thus, existing systems either resort to simple heuristic algorithms [9], [10], [15], or assign each segment an ad-hoc *utility function* and solve the simplified problem of maximizing the system-wide utility [16], [17]. These algorithms provide *no* performance guarantees on the number of on-time delivered segments and may result in playout glitches and degraded video quality. A recent work pointed out that these existing algorithms might work in live streaming systems as peers in these systems share a small scheduling window and are less sensitive to the performance of scheduling algorithms; however, they may not work well in on-demand streaming systems [11], especially when the system bandwidth resources are limited.

In this paper, we study the problem of scheduling segment transmission in on-demand P2P streaming systems. Our goal is to maximize the perceived video quality by scheduling the segment transmission so that segments that are more critical to video quality improvements are given higher priority to meet their deadlines. We first consider a general problem in which

Manuscript received July 17, 2010; revised November 10, 2010; accepted January 14, 2011. Date of publication January 28, 2011; date of current version July 20, 2011. This work was supported in part by the Natural Sciences and Engineering Research Council (NSERC) of Canada and in part by the British Columbia Innovation Council. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Thinh Nguyen.

Y. Shen and M. Hefeeda are with the School of Computing Science, Simon Fraser University, Surrey, BC V3T 0A3, Canada (e-mail: ysa57@cs.sfu.ca; mhefeeda@cs.sfu.ca).

C.-H. Hsu is with the Deutsche Telekom R&D Lab USA, Los Altos, CA 94022 USA (e-mail: cheng-hsin.hsu@telekom.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TMM.2011.2108644

different segments have different weights in terms of quality improvements to the received video. We show that this scheduling problem is NP-Complete. We then present an integer linear programming (ILP) formulation for this general problem, and we optimally solve it using an ILP solver. However, optimally solving this ILP formulation may take a long time, which is not suitable for real-time P2P streaming systems that are fairly dynamic. Thus, we propose an approximation algorithm called weighted segment scheduling (WSS), which constructs transmission schedules with performance guarantees. We derive its performance bound and implement it in an event-driven simulator as well as in a P2P prototype streaming system. The results show that the WSS algorithm leads to almost optimal perceived video quality and outperforms heuristic algorithms used in current systems. Although the WSS algorithm runs in polynomial time, it is less applicable for slower machines with limited computing power. Therefore, we consider a simplified scheduling problem in which all segments have unit weights, which is still NP-Complete. We then propose another approximation algorithm called serialized shortest transmission-time first (SSTF), which computes transmission schedules with performance guarantees. Our experimental results show that the SSTF algorithm yields performance close to the WSS algorithm and runs much faster. Thus, we recommend using the SSTF algorithm if the computational resources are limited and using the WSS algorithm when peers have enough CPU cycles.

The rest of this paper is organized as follows. Section II summarizes the related work in the literature. In Section III, we describe the considered system model and state the segment scheduling problem. We also show the hardness of the problem in the same section. We formulate and solve the general scheduling problem in Section IV and the simplified scheduling problem in Section V. Evaluation results are given in Section VI. We conclude the paper in Section VII.

II. Related Work

A measurement study on PPLive streaming system [18] reports that users suffer from long start-up delays and playout lags and suggests that better segment scheduling algorithms are required [19]. Optimally computing segment schedules to maximize the perceived video quality, however, is computationally complex [9]. Therefore, many P2P streaming systems, such as [9], [10], [15], and [20], resort to simple heuristics for segment scheduling. Pai *et al.* [10] propose to randomly schedule segment transmission, where each receiver randomly decides what segments to request from their neighbors. Zhang *et al.* [9] assume that segments with fewer potential senders are more likely to miss their deadlines and propose to schedule the segments with fewer potential senders earlier. Agarwal and Rejaie [15] describe a weighted round-robin algorithm based on senders' bandwidths. In their algorithm, the number of segments assigned to each sender is proportional to its bandwidth. In other words, senders with higher bandwidths will be assigned more segments than those with lower bandwidths. Kowalski and Hefeeda [20] propose to assign each segment to the sender that will deliver it the earliest. All these *heuristic* algorithms do not provide any performance guarantees on perceived video quality and may not perform well in on-demand streaming systems [11].

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

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

Several other works are related to the segment scheduling problem, but they do not directly solve it. Cai *et al.* [21] propose a P2P system that measures the time required to download the entire video from a number of senders and uses this information to choose senders from a large group of potential senders. Annapureddy *et al.* [11] propose using network coding to bypass the scheduling problem among small blocks belonging to the same relatively large segment. However, employing network coding may impose higher processing overhead on peers, which may require special hardware to speed up the decoding process [22] and is not easy to deploy. Several segment scheduling algorithms, such as [23], have been proposed for tree-based systems. They are, however, not applicable to mesh-based systems, in which peers have no knowledge of the global network topology.

Finally, in our earlier work [24], we introduced the WSS algorithm and evaluated its performance by simulation. In this paper, we extend that work by evaluating the performance of the WSS algorithm in both simulations and PlanetLab experiments with different settings. In addition, we propose a new approximation algorithm (SSTF), which simplifies the scheduling model and solves it efficiently.

III. Segment Transmission Scheduling Problem

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

TABLE I
LIST OF SYMBOLS USED IN THE PAPER

Symbol	Description	Symbol	Description
δ	scheduling window size	d_n	deadline of segment n
F	frame rate	$a_{n,m}$	availability of segment n on sender m
G	number of frames in each segment	$x_{n,m,t}$	variable for m to transmit n at time t
M	number of senders	$\mathbf{I}_m$	set of intervals to color
N	number of segments	P	rounding factor
T	number of time slots	U	number of variables in Eq. (1)
m_i	sender i	V	number of constraints in Eq. (1)
n_i	segment i	L	number of bits to encode Eq. (1)
t_i	transmission time for segment i	α	perceived video quality
u_i	whether segment i arrives on-time	β	continuity index
b_m	bandwidth of sender m	γ	load balancing factor
$\mathbf{Q}_m$	schedule for sender m	z	total weight of on-time segments
s_n	size of segment n	w_n	weight or value of segment n

A. System Model

We consider mesh-based (also known as swarm-based or data-driven) P2P streaming systems, which are widely deployed over the Internet. Examples of such systems include CoolStreaming [9], PPLive [18], UUSee [25], SopCast [26], and TVAnts [27]. In these systems, peers form swarms for exchanging video data. Each swarm contains a subset of the peers, and a peer may participate in multiple swarms. Data availability on peers is propagated through exchanging control messages, such as buffer maps which indicate which video segments peers currently have in their buffers. Using these buffer maps, peers *pull* video segments from each other. More specifically, a receiver simultaneously requests segments from different senders. This is done by forming a segment transmission schedule by which the receiver specifies for each sender which segments to transmit and when to transmit. In video streaming systems, the arrival times of segments are critical, as segments arriving after their decoding deadlines cannot be rendered to users and are essentially useless.

The problem addressed in this paper is to compute transmission schedules for receivers in order to optimize their perceived video quality. Transmission schedules are computed for recurring sliding time windows, whose lengths are in the order of seconds. Our problem formulation and solution employ a *realistic* model for P2P streaming systems. Thus, our proposed algorithm can readily be implemented in current mesh-based P2P streaming systems to improve their performance. Particularly, we assume that P2P streaming systems are highly dynamic and peers will join and leave frequently. Thus, we design our algorithms to be lightweight and can be invoked whenever such events occur in order to quickly recompute a new transmission schedule. In addition, the dynamic propagation and replication of video segments in the P2P streaming system can easily be handled by our algorithms. This is because segment propagation will trigger peers to update their buffer maps to reflect the availability of the newly acquired segments. When these buffer maps are exchanged among peers in control messages, our scheduling algorithms will account for the new segments in computing new transmission schedules.

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

B. Problem Statement and Hardness

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

To generate feasible schedules, the receiver monitors its senders in terms of segment availability and uploading bandwidth. We let $a_{n,m}$ be the availability of segment $n(1 \leq n \leq N)$ on sender $m(1 \leq m \leq M)$. The receiver sets $a_{n,m} = 1$ if sender m has a copy of segment n and $a_{n,m} = 0$ otherwise. We let b_m Kbps be the uploading bandwidth of sender m. With the segment availability and senders' bandwidths, the receiver composes a segment schedule for a sliding window

of δ seconds, where δ is a system parameter. The resulting schedule is sent to senders, and senders transmit segments following the schedule. Furthermore, different segments have different impacts on video quality improvements. Let w_n be the *weight* or the *value* of segment n, which represents the quality improvement brought by this segment. We consider a general problem in which the definition of w_n is determined by P2P systems.

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

Problem 1 (Segment Transmission Scheduling): We consider the segment transmission scheduling problem of video sequences in P2P streaming systems. The problem is to construct an optimal transmission schedule $\mathbf{Q} = \{\langle m_i, n_i, t_i\rangle,\ 1 \le i \le Q\}$ for a sliding window of δ seconds, where m_i indicates the sender, n_i represents the segment, t_i is the transmission time, and Q is the number of segments in the sliding window. A segment n_i is said to be on-time if and only if $t_i + s_{n_i}/b_{m_i} \le d_{n_i}$. We let $u_n = 1$ if segment n arrives on-time at the receiver from any of its senders and $u_n = 0$ otherwise. The objective is to maximize the perceived video quality, which is the sum of weights (w_n) of all segments that arrive on-time.

The next theorem states that solving this segment scheduling problem is computationally expensive.

Theorem 1 (Hardness): The segment transmission scheduling problem defined in Problem 1 is NP-Complete. This is true even when all segments are equally important, i.e., $w_n = 1$ for all $1 \le n \le N$.

The proof of this theorem is omitted due to space limitations; it can be found in [28]. The hardness of the segment scheduling problem is also mentioned in [9].

We refer to Problem 1 as *weighted*, or general segment scheduling problem and to the version with unit w_n values as *unweighted*, or simplified segment scheduling problem. We solve the former problem in Section IV and the latter one in Section V.

IV. Solution for the General Problem

In this section, we consider the general scheduling problem and we use peak signal-to-noise ratio (PSNR) to define w_n as it is widely used in multimedia systems [29, Sec. 1.5.5]. More precisely, we let w_n be the average video quality in PSNR of segment n. The w_n weights (or values) are typically pre-computed by video coders and inserted into coded streams as *meta data*. That is, they are not computed at streaming time. The computation of PSNR values can be done empirically for higher accuracy or by some R-D models for lower overhead.

A. Formulation

We formulate the segment scheduling problem as a time-indexed integer linear programming (ILP) problem. Time-indexed formulations discretize the time axis into T time slots, where T is large so that the time slots are fine enough to represent any feasible schedule without reducing the value of the objective function. We let $x_{n,m,t}$ be a 0–1 variable for each $n = 1, 2, \ldots, N$, $m = 1, 2, \ldots, M$, and $t = 1, 2, \ldots, T$, where $x_{n,m,t} = 1$ if segment n is scheduled to be transmitted by sender m at time t and $x_{n,m,t} = 0$ otherwise. We note that, according to the definition, while the transmission of a segment often spans over several time slots, only the *first* time slot has an x value of 1.

We formulate the considered segment scheduling problem as

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

$$\text{s.t. } x_{\hat{n},\hat{m},\hat{t}} \le a_{\hat{n},\hat{m}} \tag{1b}$$

$$\sum_{n=1}^{N} \sum_{t=\hat{t}-s_n/b_{\hat{m}}+1}^{\hat{t}} x_{n,\hat{m},t} \le 1 \tag{1c}$$

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

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

In this formulation, the objective function in (1a) is to maximize the sum of weights of all on-time segments, where the three summations iterate through all segments, senders, and time slots, respectively. Note that the last summation stops at time $d_n - s_n/b_m$ if $d_n - s_n/b_m < T$, or T otherwise, because scheduling segment n *after* that time results in a late segment, which cannot improve video quality and in each scheduling period, we are only interested in scheduling segments within the scheduling window. The constraint in (1b) makes sure that we always schedule a segment to a sender who holds a copy of it, as it prevents the combination of $x_{\hat{n},\hat{m},\hat{t}} = 1$ and $a_{\hat{n},\hat{m}} = 0$ for all $\hat{t} = 1, 2, \ldots, T$. In (1c), observe that any segment n scheduled on sender $\hat{m}$ between time $\hat{t} - s_n/b_{\hat{m}} + 1$ and $\hat{t}$ would *occupy* the time slot $\hat{t}$ as transmitting segment n takes time $s_n/b_{\hat{m}}$. Therefore, by considering all these segments, the constraint in (1c) ensures that at most one segment is scheduled on each sender at any time $\hat{t}$. Last, the constraint in (1d) prevents segments from being scheduled on more than one sender.

Optimal Algorithm: To get optimal segment schedules, we can solve the formulation in (1) using any ILP solver. In this paper, we use CPLEX [30] for this purpose. We refer to this approach as the OPT algorithm and use it as a benchmark to assess the performance of all other algorithms. Notice that since the optimal solution may take a long time to compute, we solve all ILP problems *offline* in the OPT algorithm.

B. Overview of the Proposed Algorithm

Solving ILP problems is computationally expensive and may not be possible in real time. Therefore, we develop an efficient approximation algorithm in the following. Our algorithm is based on the linear programming (LP) relaxation of the ILP formulation in (1). The LP relaxed formulation allows any $x_{\hat{n},\hat{m},\hat{t}}$ in (1e) to take fractional values, where $0 \le x_{\hat{n},\hat{m},\hat{t}} \le 1$. This LP relaxed formulation can be optimally solved using efficient LP solvers that implement Simplex or interior point methods (IPMs). We use $\bar{x}_{\hat{n},\hat{m},\hat{t}}$ to denote the *fractional schedule* produced by an LP solver. We notice that fractional schedules of the LP relaxed formulation are not feasible to the original scheduling problem. This is because, in the LP relaxed

formulation, the constraints in (1c) and (1d) are interpreted in a different way: the constraint in (1c) makes sure that the fractions of all segments scheduled on each sender sum to at most one at any time and the constraint in (1d) ensures that fractions of each segment scheduled on all senders sum to at most one at any time.

To compute a schedule for the original scheduling problem, we propose a rounding algorithm to convert fractional schedules into integral *feasible* schedules, albeit with a small approximation factor. We first explain how the rounding algorithm handles fractional schedules on single sender m for $m = 1, 2, \ldots, M$. We then expand it to the general case of M senders. For a specific sender m, the rounding algorithm consists of two steps: 1) it transforms the *fractional schedule* into several feasible *integral schedules* and 2) it selects the best schedule out of all integral schedules. More precisely, the rounding algorithm first rounds the fractional schedule $\bar{x}_{\hat{n},m,\hat{t}}$ for all $\hat{n} = 1,2,\ldots,N$ and $\hat{t} = 1,2,\ldots,T$ to multiples of $1/P$, where $P = (TN)^2$. This is achieved by creating $\lfloor \bar{x}_{\hat{n},m,\hat{t}} \times P \rfloor$ copies of *time intervals* $[\hat{t}, \hat{t} + s_{\hat{n}}/b_m]$ for each positive $\bar{x}_{\hat{n},m,\hat{t}}$. These time intervals are then put in the set $\mathbf{I}_m$. Next, we color the intervals in $\mathbf{I}_m$ with minimum number of colors, so that: 1) two intervals overlapping in time have different colors and 2) two intervals of the same segment have different colors. This can be done by first sorting all intervals on their starting times and then sequentially coloring them in that order. Once the coloring is done, intervals with the same color have two nice properties, i.e., they: 1) never overlap in time and 2) are not associated with the same segment. Therefore, we can construct an integral schedule using all intervals that have the same color. The coloring scheme results in several feasible schedules. We then compute the objective function value of each feasible schedule and choose the schedule with the largest objective function value. We let this schedule be $\mathbf{Q}_m = \{\langle m, \hat{n}, \hat{t} \rangle\}$, which indicates that sender m should start transmitting segment $\hat{n}$ at time $\hat{t}$.

For the general case of M senders, the rounding algorithm sequentially schedules segments for all senders. More specifically, for each sender m, the rounding algorithm sets all $\bar{x}_{n,m,\hat{t}} = 0$ for all $\hat{t} = 1,2,\ldots,T$, *if* segment n has been scheduled on any sender $\bar{m}$, where $\bar{m} < m$. This is to avoid scheduling a segment to multiple senders, which violates the constraint in (1d). Once the feasible schedule for sender $m (1 \le m \le M-1)$ is derived, the rounding algorithm goes on to construct schedules on sender $m+1$. It stops after iterating through all senders and returns the segment schedule $\mathbf{Q}_m$, for all $m = 1,2,\ldots,M$. Since our proposed algorithm takes user-specified weights, we call it weighted segment scheduling (WSS) algorithm.

Fig. 1 shows a high-level pseudocode of the WSS algorithm. It solves the LP relaxed formulation in line 3. The rounding is sequentially done using the for-loop between lines 4 and 15. The foreach-loop between lines 6 and 8 builds the set $\mathbf{I}_m$ of time intervals for sender m based on its fractional schedule. Line 9 colors $\mathbf{I}_m$ using as few colors as possible. Line 10 builds a set of feasible integral schedules and line 11 picks the schedule $\mathbf{Q}_m$ that leads to the highest objective function value z. The for-loop between lines 12 and 14 prevents segments from being scheduled to multiple senders. The algorithm returns $\mathbf{Q}_1, \mathbf{Q}_2, \ldots, \mathbf{Q}_M$ in line 16.

WSS: Weighted Segment Scheduling

1. **let** $\mathbf{Q}_m = \varnothing$, where $m = 1, 2 \ldots, M$
2. **let** $P = (TN)^2$
3. **compute** optimal $\bar{x}_{\hat{m},\hat{n},\hat{t}}$ for the relaxed Eq. (1)
4. **for** $m = 1$ to M // consider senders sequentially
5. **let** $\mathbf{I}_m = \varnothing$ // time intervals
6. **foreach** positive $\bar{x}_{m,\hat{n},\hat{t}}$
7. **insert** $\lfloor \bar{x}_{m,\hat{n},\hat{t}} \times P \rfloor$ copies of interval $[\hat{t}, \hat{t} + \frac{s_{\hat{n}}}{b_m}]$ to $\mathbf{I}_m$
8. **endfor**
9. **color** intervals in $\mathbf{I}_m$ using fewest colors, so that two
9. intervals overlapping in time or associated with the
9. same segment have different colors
10. **construct** a feasible schedule for each color
11. **let** $\mathbf{Q}_m$ be feasible schedule that results in highest
11. objective function value z
12. **for** $\hat{m} = m + 1$ to M // mark as scheduled
13. **let** $\bar{x}_{\hat{m},\hat{n},\hat{t}} = 0$
14. **endfor**
15. **endfor**
16. **return** $\mathbf{Q}_1, \mathbf{Q}_2, \ldots, \mathbf{Q}_M$.

Fig. 1. Proposed approximation algorithm for the weighted segment transmission scheduling problem.

C. Approximation Factor and Complexity

We analyze the performance of the WSS algorithm in two steps. We first analyze the problem with a single sender and then extend the analysis to multiple senders. In the next lemma, we derive the approximation factor of the WSS algorithm with one sender. In the proof, we first show that the floor function in line 7 incurs negligible drops on the objective function value. We then show that the number of colors required in line 9 is bounded. Therefore, we only have a *small* number of feasible schedules and at least one of them leads to an objective value higher than half of the optimum.

Lemma 1: The WSS algorithm in Fig. 1 has an approximation factor of 2, when there is only one sender.

Proof: Let z^* be the optimal objective function value for the relaxed LP problem. It is easy to see that z^* is an upbound on the objective value of the original integral solution. In line 7, each $\bar{x}$ is rounded to a multiple of $1/P$. This means that each $x_n^* w_n$ in the objective function (1a) is reduced by at most $1/P$ by the rounding. Let $w_{n'}$ be the maximal weight among all scheduled segments. Since we have (TN) $\bar{x}$ variables, the value of z^* drops at most $(TN)w_{n'}/P = w_{n'}/(TN)$. Observe that, since scheduling only segment n' results in a feasible schedule with a sum of weights $w_{n'}$, we have $z^* \ge w_{n'}$. Thus, the rounding in line 7 reduces the z^* value by at most a factor of $1/(TN)$, which is negligible as the number of time slots T is large.

From the formulation in (1), the fractional schedule produced by the LP solver has the following property: the number of overlapping intervals is at most P and the number of intervals belonging to the same segment is at most P. Following the coloring strategy in line 9, the intervals can be colored with at most $C = 1 + (P-1) + (P-1) = 2P - 1$ different colors. Furthermore, line 7 indicates that $z^* = \sum_{c=1}^{C} \hat{w}_c / P$, where $\hat{w}_c (c = 1,2,\ldots,C)$ is the objective function value of the feasible schedule derived from color c. Defining $\alpha_c = 1/P$ for each $c = 1,2,\ldots,C$, we have

$$z^* = \sum_{c=1}^{C} \hat{w}_c \alpha_c. \tag{2}$$

Moreover, we observe that

$$\sum_{c=1}^{C} \alpha_c = \sum_{c=1}^{2P-1} \left(\frac{1}{P}\right) = 2 - \left(\frac{1}{P}\right) < 2. \tag{3}$$

With (2) and (3), we claim that there exists a color c^* such that $\hat{w}_{c^*} \geq 1/2z^*$. This claim can be easily proved by contradiction. Combining the fact that z^* is an upbound of the original integral solution, we conclude that the feasible schedule identified by line 11 achieves approximation factor 2, when $M = 1$. ∎

Next, we extend the above lemma to the general case of multiple senders.

Theorem 2 (Approximation Factor): The WSS algorithm in Fig. 1 achieves approximation factor of 3, when there are multiple senders.

Proof: For $m = 1, 2, \ldots, M$, we let $\mathbf{R}_m$ be the fractional schedule of sender m produced in line 3 and $\mathbf{Q}_m$ be the integral schedule of sender m returned in line 16. We also define $\mathbf{R}'_m$ as the fractional schedule of sender m after discarding all segment scheduled in $\mathbf{Q}_1, \mathbf{Q}_2, \ldots, \mathbf{Q}_{m-1}$. We use $w(\cdot)$ to denote the objective function value of $\mathbf{R}_m$, $\mathbf{Q}_m$, and $\mathbf{R}'_m$.

Following Lemma 1, we have $w(\mathbf{Q}_m) \geq (1/2)w(\mathbf{R}'_m)$, for all $m = 1, 2, \ldots, M$. Since $\mathbf{Q}_m$ are mutually disjoint and $\mathbf{R}'_m$ are mutually disjoint, we have

$$\sum_{m=1}^{M} w(\mathbf{Q}_m) \geq \frac{1}{2} \sum_{m=1}^{M} w(\mathbf{R}'_m). \tag{4}$$

Since every scheduled segment is marked by the for-loop between lines 12 and 14, we write

$$\sum_{m=1}^{M} w(\mathbf{R}'_m) \geq \sum_{m=1}^{M} w(\mathbf{R}_m) - \sum_{m=1}^{M} w(\mathbf{Q}_m). \tag{5}$$

Combining (4) and (5) gives

$$\sum_{m=1}^{M} w(\mathbf{Q}_m) \geq \frac{1}{2} \sum_{m=1}^{M} w(\mathbf{R}_m) - \frac{1}{2} \sum_{m=1}^{M} w(\mathbf{Q}_m).$$

Rearranging this inequality, we get

$$\sum_{m=1}^{M} w(\mathbf{Q}_m) \geq \left(\frac{1}{3}\right) \sum_{m=1}^{M} w(\mathbf{R}_m)$$

which yields an approximation factor of 3. ∎

We show that the proposed WSS algorithm is a polynomial time algorithm in the next theorem.

Theorem 3 (Complexity): The WSS algorithm proposed in Fig. 1 runs in polynomial time, i.e., it terminates in $O(3UP + M(2P-1) + MU + U^{1.5}V^2L)$ arithmetic operations, where $P = (TN)^2$ is the rounding factor, $U = MNT$ is the number of variables of the formulation in (1), $V = MNT + MT + T$ is the number of constraints in this formulation, and $L = 8U + UV + V$ is the number of total bits required to encode this formulation.

Proof: Notice that there are $U = MNT$ variables in the objective function shown in (1a). Furthermore, there are MNT constraints in (1b), MT constraints in (1c), and N constraints in (1d), and thus, we have $V = MNT + MT + N$ total constraints.

The time complexity before line 4 can be bounded by $O(\sqrt{U}LUV^2)$ following the complexity results of IPMs in the literature [31, Sec. 4.6], where L is the number of total bits required to write the formulation. L can be derived as follows. In (1a), we consider the weights w_n for all $n = 1, 2, \ldots, N$ are represented as 8-bit floating point values, which take $8U$ bits. In (1b)–(1d), we have UV coefficients on the left of the inequalities and U on the right. Observe that all these coefficients can be encoded in a single bit, which means encoding the constraints requires $UV + V$ bits. Combining the objective function with constraints, we get $L = 8U + UV + V$.

Next, we compute the number of operations required by each iteration of the for-loop between lines 4 and 15. The foreach-loop in lines 6–8 creates at most $O(NTP)$ intervals. The coloring in line 9 can be done in a single-pass scan on the intervals in $\mathbf{I}_m$, which takes $O(NTP)$ operations. Constructing feasible solutions in line 10 requires $O(NTP)$ and finding the solution with the highest objective function value consumes $O(NTP) + O(2P - 1)$. The for-loop between lines 12 and 14 takes $O(MNT)$ operations. This leads to $O(3NTP + 2P + MNT - 1)$ operations in each iteration between lines 4–15 and $O(3UP + M(2P-1) + MU)$ for the whole rounding algorithm. Combining this with $O(\sqrt{U}LUV^2)$ yields the theorem. ∎

Finally, we mention that a similar LP relaxation method was used in designing other scheduling algorithms with different objectives, such as minimizing the task completion time [32], [33] and maximizing total weight of tasks completed by their due dates [34].

V. Solution for Unweighted Problem

The WSS algorithm proposed in Section IV may be computationally demanding for P2P streaming systems with limited computational resources. Therefore, it is desirable to have a simpler algorithm for those P2P streaming systems. We consider a simplification of the general scheduling in this section, where all segments have the same weight (w_n) value. The objective of maximizing the perceived video quality then becomes maximizing the number of on-time segments. This simplification can reduce both computational and communication overheads.

A. Formulation

The unweighted problem can be formulated in various ways. For example, we can still use a time-indexed formulation as in (1). However, doing so may prevent us from utilizing some unique properties of the unweighted scheduling problem for designing better algorithms. More specifically, the unweighted problem has a nice property: Assuming $\hat{N}$ segments can be optimally scheduled on sender m (i.e., all of them arrive on-time), scheduling these $\hat{N}$ segments in ascending order of their deadlines is one of the optimal schedules. We formally describe this property in the next theorem.

Theorem 4: For $\hat{N}$ segments that can be scheduled on sender m, the schedule $\mathbf{Q}^*_m = \{\langle m, n, t_n\rangle,\ 1 \leq n \leq \hat{\mathrm{N}}$ and $\mathrm{t}_n = \sum_{i=1}^{n-1} \mathrm{s}_i/\mathrm{b}_m\}$ is an optimal schedule, where segments are sorted in ascending order on their deadlines.

Proof: Let $\bar{\mathbf{Q}}_m$ be any optimal solution that maximizes the number of on-time arrival segments from sender m. We first

eliminate any idle time in $\bar{\mathbf{Q}}_m$ by shifting all segments to their left whenever possible. This preprocessing step does not affect the optimality of $\bar{\mathbf{Q}}_m$ because all on-time segments complete not later than before. Next, if segments in $\bar{\mathbf{Q}}_m$ are sorted in ascending order on deadlines, letting $\mathbf{Q}^*_m = \bar{\mathbf{Q}}_m$ yields the theorem. Otherwise, we recursively apply the transformation described below.

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

This theorem shows that the segment assignment determines the optimality of segment schedules. This enables us to transform the unweighted segment transmission scheduling problem into an *assignment* problem, which is less complicated. Before presenting the new formulation, we mention that similar theorems have been used to formulate machine scheduling problems, e.g., in [35, Sec. 3.3] and [36].

Theorem 4 states that sorting segments in ascending order on deadlines does not affect the existence of optimal schedules. Therefore, without loss of generality, we assume $d_1 \leq d_2 \leq \cdots \leq d_N$. Otherwise, we sort and relabel segments. We let $x_{n,m}$ $(1 \leq n \leq N \text{ and } 1 \leq m \leq M)$ be the *assignment variable*, where $x_{n,m} = 1$ if segment n is assigned to sender m and $x_{n,m} = 0$ otherwise. We then formulate the unweighted problem as follows:

$$z = \max \quad \sum_{n=1}^{N}\sum_{m=1}^{M} x_{n,m} \tag{6a}$$

$$\text{s.t.} \quad x_{\hat{n},\hat{m}} \leq a_{\hat{n},\hat{m}} \tag{6b}$$

$$\sum_{n=1}^{\hat{n}} \left(\frac{s_n}{b_{\hat{m}}}\right) x_{n,\hat{m}} \leq d_{\hat{n}} \tag{6c}$$

$$\sum_{m=1}^{M} x_{\hat{n},m} \leq 1 \tag{6d}$$

$$x_{\hat{n},\hat{m}} \in \{0,1\},$$
$$\forall \hat{n} = 1,2,\ldots,N \text{ and } \hat{m} = 1,2,\ldots,M. \tag{6e}$$

In this formulation, the objective function in (6a) is to maximize the number of on-time arrival segments. The first constraint in (6b) ensures that we always schedule a segment to a sender that holds a copy of it, as it prevents the combination of $x_{n,m} = 1$ and $a_{n,m} = 0$. The second constraint in (6c) computes the accumulated transmission time of sender $\hat{m}$ up to and including segment $\hat{n}$ and checks whether segment $\hat{n}$ is complete before its deadline. The third constraint in (6d) avoids assigning a segment to more than one sender. Notice that, compared to time-indexed formulation in (1), this formulation utilizes the unique property (Theorem 4) of the unweighted scheduling problem and has fewer variables and constraints.

SSTF: Serialized Shortest Transmission-time First

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

Fig. 2. Proposed approximation algorithm for the unweighted segment transmission scheduling problem.

B. Overview of the Proposed Algorithm

The idea of our algorithm can be described as follows. First, for any given unweighted problem, it considers each sender sequentially. That is, the algorithm assigns sender m as many segments as possible, before it assigns the remaining segments to sender $m+1$. Second, for sender m, the algorithm repeatedly schedules the segment with the shortest transmission time first among all the segments that: 1) have not been scheduled to any senders and 2) can be transmitted entirely by sender m before their deadlines. Since the proposed algorithm sequentially schedules senders $1,2,\ldots,M$, we call it serialized shortest transmission-time first (SSTF) algorithm.

Fig. 2 presents a high-level pseudocode of the SSTF algorithm. This algorithm puts all segments in $\bar{\mathbf{N}}$ in line 2 and sorts these segments in ascending order with respect to segment size in line 3. Sorting segments allows us to efficiently locate the segment with the *shortest* transmission time from any sender m. This is because the bandwidth b_m is independent from which segment to send, and thus, the transmission time of different segments on the same sender is proportional to the segment size. The algorithm then considers each sender sequentially in the for-loop from lines 4 to 11. The foreach-loop between lines 5 and 11 iterates through $\bar{\mathbf{N}}$ in ascending order with respect to segment size and the if-statement in line 7 identifies possible assignment by checking: 1) is segment n available on sender m? and 2) can segment n been transmitted by sender m arrive on-time? If both conditions hold, the algorithm schedules segment n on sender m by moving that segment from $\bar{\mathbf{N}}$ to $\mathbf{Q}_m$. It also updates t, which represents the amount of time on sender m that has been consumed. Finally, the algorithm returns the segment transmission schedule in line 12.

C. Approximation Factor and Complexity

We analyze the approximation factor of the SSTF algorithm in the next theorem.

Theorem 5 (Approximation Factor): The SSTF algorithm given in Fig. 2 returns a segment transmission schedule that yields a number of on-time segments that is at most a factor of 2 compared to the optimal solution.

Proof: We first consider a specific sender m in the for-loop between lines 4 and 11. We let $\mathbf{S_m}$ be the set of segments for

TABLE II
LIST OF VIDEO PARAMETERS USED IN THE PAPER

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

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

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

Next, we show that the SSTF algorithm is efficient.

Theorem 6 (Time Complexity): The SSTF algorithm given in Fig. 2 runs in time $O(MN + N \log N)$, where M is the number of senders and N is the number of segments.

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

We notice that M and N are typically small values. This is because M is the number of potential senders for a given receiver, which is in the order of tens of senders and N is the number of segments in each scheduling window, which is also in the order of a few tens of segments. Thus, our algorithm can easily run in real time and be invoked frequently to handle the high dynamics of P2P streaming systems.

VI. EVALUATION

In this section, we evaluate the proposed algorithms and compare them against others in the literature using extensive simulations and experiments on the PlanetLab wide-area network testbed. We use a wide range of realistic network and peer parameters and evaluate several performance metrics using actual video traces.

A. Simulation Setup

We have implemented an event-driven simulator in Java to evaluate the performance of the proposed segment scheduling algorithms and compare them against other algorithms. Five scheduling algorithms are implemented in this simulator: RF [9], MC [16], SSTF, WSS, and OPT. The RF algorithm implements the rarest first algorithm. It schedules the segment with the fewest potential senders first and among multiple senders, the one with highest bandwidth and enough available time first. RF is a fairly common algorithm used in several widely deployed P2P systems, such as CoolStreaming [9] and PPLive [18]. MC is a quite sophisticated algorithm that has been proposed in the literature. The MC algorithm is based on an ILP formulation, which is converted into a min-cost flow problem and solved by combinatorial algorithms. While we employ the same utility function defined in the evaluation section of [16], the original algorithm can only schedule transmission of fixed-size blocks. We, therefore, extend that algorithm to support variable-size segments by: 1) dividing each segment into blocks, 2) solving the block transmission problem using their algorithm, and 3) for each segment, trying to schedule it on the sender which has been assigned the most number of blocks. If that sender has used up all the bandwidth, we then try to schedule it on the sender which has been assigned the second most number of blocks and so on. SSTF is the implementation of our unweighted approximation algorithm and WSS is the implementation of our weighted approximation algorithm. In the OPT algorithm, we directly solve the ILP formulation in (1) with the CPLEX ILP solver. In particular, the CPLEX package provides a set of Java class libraries that allow us to specify and solve our problems using Java syntax through Java native interface (JNI).

We choose two high-quality videos with different characteristics from the Arizona State University video trace library [37] to analyze the performance of our algorithms. Table II summarizes the main parameters of the videos. We use the PSNR as the perceived video quality metric and as the segment weight (w_n) in our simulation. The simulator puts every GOP into one segment, so that segment n has a decoding deadline of $G(n-1)/F$, where $n = 1, 2, \ldots, N$. Packing each GOP into a scheduling unit ensures that every received segment can be decoded independently at the receiver, since dependency among video frames is restricted within a GOP. Some previous algorithms, like the

TABLE III
PEER UPLOADING BANDWIDTH DISTRIBUTION

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

original MC algorithm proposed in [16], divide videos into fixed size blocks as scheduling units. This can cause segments not to be decodable, because even if a very small part of a segment is not received on-time, the whole segment cannot be decoded, which results in degraded video quality.

We simulate a dynamic system with a total number of 2000 peers. We initially pre-deploy the video sequences on only 1% of the peers chosen randomly, which form the initial seeding peers. We run the simulation for 24 h for each algorithm. Individual peers dynamically join and leave a swarm at different times during the streaming of a video sequence. The joining and leaving times are randomly chosen from the whole simulation time period following a uniform distribution. Upon joining a swarm, each peer is instructed to stream the video sequence. We also simulate dynamic replication of video segments as peers can upload segments as soon as they have downloaded those segments from other peers. We consider a peer matching service that randomly provides each new peer up to ten senders. Each new peer then connects to these senders, runs the scheduling algorithm, and requests segments following the scheduling results. We set the scheduling window to be 10 s in simulations.

The simulator determines each sender's uploading bandwidth following the distribution given in Table III. This bandwidth distribution is proposed in a paper [38] based on various measurement studies on both corporate and residential users. We note that peers do *not* contribute all their bandwidth to P2P streaming, because doing so slows down other Internet applications such as e-mail and Web. The *contributed bandwidth* of each class of peers is also given in this table as recommended in [38]. With the randomly chosen bandwidth, the simulator fairly distributes available bandwidth among all connections and predicts the transmission time for each packet accordingly. Finally, in the OPT and WSS algorithms, the system parameter T is set to be 100, which means the time slots are 100 ms long in our formulation.

We run the simulator independently for each considered algorithm on a commodity PC, with a 2.66-GHz Intel CPU and 8 GB of memory. We define three performance metrics: the average perceived video quality α, the continuity index β, and the load balancing factor γ. The average perceived video quality is computed by assuming that all late segments result in zero PSNR and defining $\alpha = \sum_{n=1}^{N} w_n u_n / N$, where w_n is the average perceived video quality of video frames in segment n. In our setup, it is the average PSNR value of the segment. We define the continuity index as the number of segments that arrive by their decoding deadlines over the total the number of segments in the video. That is, $\beta = \sum_{n=1}^{N} u_n / N$. Last, we define the load of sender m as its uploading bandwidth utilization, which is $\left(\sum_{n \in \mathbf{Q}_m} s_n / \delta\right) / b_m$, where $\sum_{n \in \mathbf{Q}_m} s_n$ accounts for the total size of all segments scheduled on that sender in one scheduling period and δ is the length of scheduling window. The load balancing factor γ is then computed as the standard deviation of loads for all scheduling periods on that sender. Similar performance metrics are used in other works in the literature, such as [9] and [39]. We note that we compute all performance metrics using the successful transmissions.

B. Simulation Results

1) Overall Comparison: For each simulation, we calculate the average performance for each peer across all the scheduling periods. We iterate through all peers and compute the cumulative distribution function (CDF) curves of each performance metric. We repeat the same computation for each scheduling algorithm. The results are summarized in Fig. 3.

Fig. 3(a) plots video quality achieved by different algorithms. This figure shows that the WSS and SSTF algorithms substantially outperform the other two heuristic algorithms RF and MC and they stay very close to the OPT. First, for the lowest 5% of the peers in terms of perceived video quality, the WSS and SSTF algorithms achieve an average perceived video quality of at least 27 dB and 28 dB, respectively, where the optimal solution achieves 29 dB, while the MC and RF algorithm can only achieve 16 dB and 17 dB, respectively. Second, both the WSS and SSTF algorithms achieve more uniform and higher video quality for all peers compared to the two heuristic algorithms. This is shown by the concentration of the CDF curves between 25 dB and 35 dB by the two algorithms, while the CDF curves of the other two heuristic algorithms are spread over larger ranges, from 15 dB to up to 30 dB.

We then report the continuity index in Fig. 3(b). This figure illustrates that the WSS and SSTF algorithms result in much higher continuity index than the RF and MC algorithms. For example, more than 98% of the peers observe a continuity index of at least 60% using the proposed WSS and SSTF algorithm, while less than 40% of the peers observe that continuity index for the RF and MC algorithms. This means employing the WSS and SSTF algorithms significantly reduces the playout glitches at receivers. Fig. 3 clearly shows that the proposed WSS and SSTF algorithms result in much higher and smoother video streaming quality, compared to the MC and RF algorithms.

We next plot the load balancing factor in Fig. 3(c). Excessive load balancing factor may slow down some senders, which could discourage users from contributing to the P2P network. This figure illustrates that the WSS algorithm achieves at most 29% deviation and thus distributes the transmission load fairly among senders. The SSTF algorithm leads to a little bit higher load balancing factor, but still within 35% deviation, and at the same time, they produce better video quality and higher continuity index as shown in Fig. 3(a) and (b). Meanwhile, the other two algorithms result in similar diversity in the load imposed on peers but worse video quality as shown in the figures.

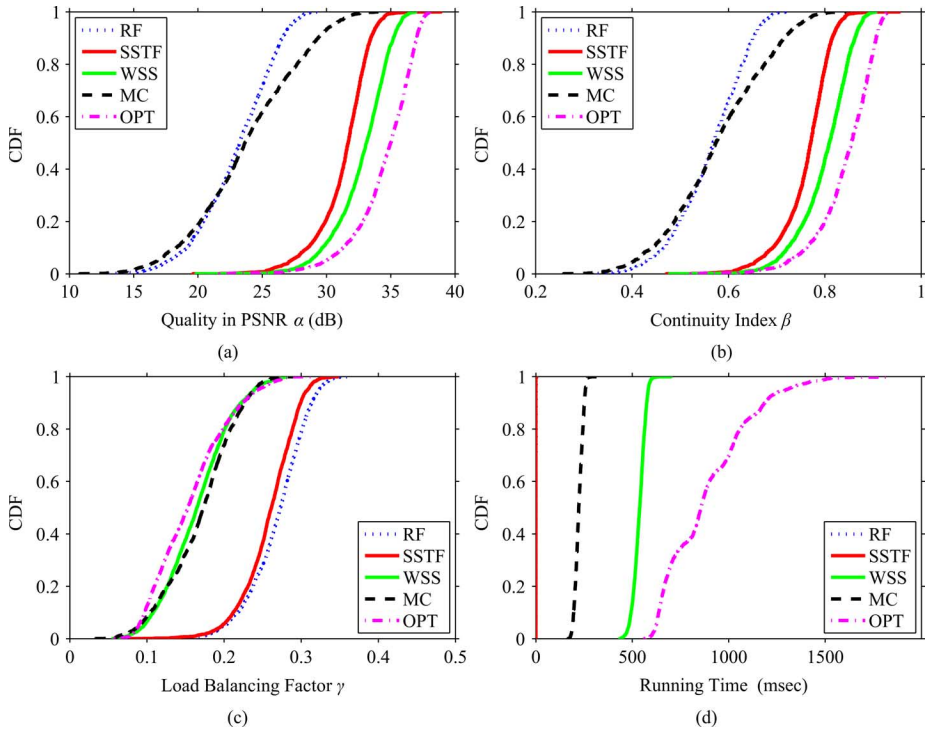


Fig. 3. Overall comparison among different algorithms. Sample results from `Terminator 2`. Note: the running time curves for the SSTF and RF algorithms are not visible in the figure because they are very close to 0. (a) Video quality. (b) Continuity index. (c) Load balance. (d) Running time.

Last, we plot the average running time across all the scheduling periods on each sender in Fig. 3(d). We can see that the SSTF and RF algorithms run much faster than the others. Notice that the running time curves for the SSTF and RF algorithms are not visible in the figure because they are very close to 0. On the other hand, except for the OPT algorithm, all the other algorithms can run in the scale of milliseconds, which is small compared to the scheduling window. For the MC algorithm, although the min-cost flow problem can be solved in polynomial time, converting the ILP problems to min-cost flow problems as proposed in [16] can result in large number of nodes in the model; thus, it runs much slower than the SSTF and RF algorithms. Notice that for the OPT algorithm, we set a bound on the maximum number of iterations for the ILP solver to prevent it from taking too long to solve the ILP problem, and we do not consider such cases in the figure.

2) Segment-Level Comparison: Next we iterate through all the video frames and compute the average PSNR in each segment for every receiver. We then compute the average video quality across all peers. We repeat the computation for all scheduling algorithms. To clearly show the results of the whole video sequence, we aggregate every ten segments and compute their average PSNR. Fig. 4(a) plots sample results from `Terminator 2`. The figure shows that our WSS and SSTF algorithms perform much better than the other two heuristic algorithms throughout the whole video period. We notice that there are a number of short-period drops in the whole sequence. These are caused by the missed or late segments. The figure shows that the number of missed segments (quality drops) is much smaller in our algorithms compared to the other two. To clarify it further, we zoom in several regions of the Fig. 4(a) to individual segment level and extract three intervals from the whole sequence: The first 50 segments, 50 segments from the middle and 50 segments from the last part of the video and plot them in Fig. 4(b)–(d), respectively. These figures clearly show that the WSS and SSTF algorithms result in higher perceived video quality. The RF and MC algorithms lead to too many quality drops, which degrade user experience.

C. Prototype Implementation and Experimental Setup

To evaluate the proposed algorithms in a real system, we have developed a prototype P2P streaming system and deployed it on PlanetLab [40]. Fig. 5 shows the high level diagram of our prototype system implementation. The system consists of one tracker and a set of peers interested in streaming a video file. The tracker is used to coordinate all peers. It keeps a list of all currently active peers in the peer manager module and performs peer matching in the peer matching module when a peer asks for a list of neighbors. There are several peer matching algorithms proposed in the literature, such as [41] and [42]. Since in this paper we focus on the scheduling part of the system, we use the random peer matching algorithm for its simplicity. That is, the tracker randomly selects a peers from the peer list and returns them to the requesting peer.

On the peer side, each peer contains three main modules: the peer manager module that establishes and maintains connections with other peers, the buffer map manager module that keeps track of the availability of segments on this peer, and the scheduler module that does segment transmission scheduling based on its neighborhood information and buffer map availability information.

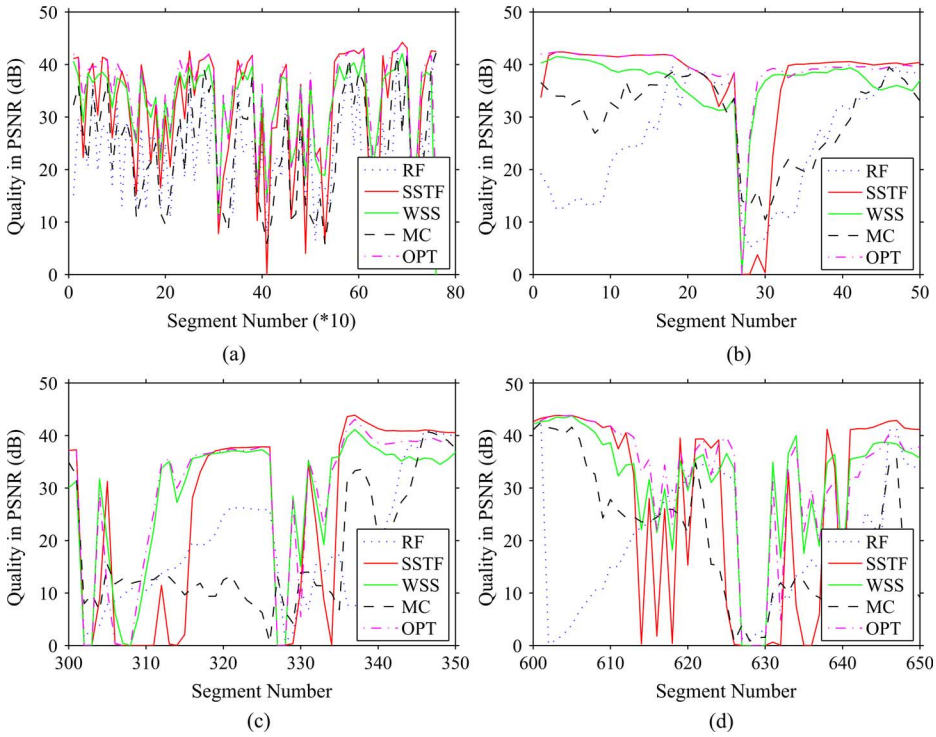


Fig. 4. Segment-level quality comparison among different algorithms. Sample results from `Terminator 2`. (a) Over the whole video sequence. (b) Sample period (i). (c) Sample period (ii). (d) Sample period (iii).

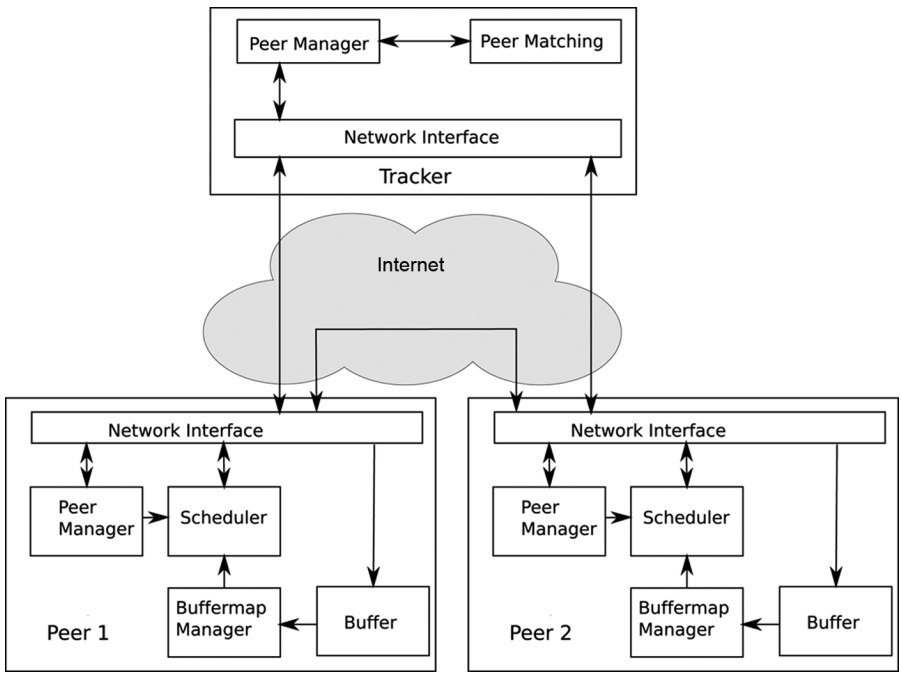


Fig. 5. High-level diagram for the prototype of our P2P streaming system implementation.

The system works as follows.

1) The tracker starts by listening on a port that is known to all the peers.
2) All peers start by first connecting to the tracker in time randomly distributed across the whole experiment. We assume the first few peers connected to the tracker have the whole video sequence. The rest of peers get a list of neighbors from the peer matching algorithm once they connect to the tracker and start exchanging data with their neighbors.
3) On each peer, for each scheduling period, the scheduler computes a segment transmission schedule based on the senders and segments information and then the peer requests data from its senders accordingly. The received data are put into the buffer for playout and the buffer map is then

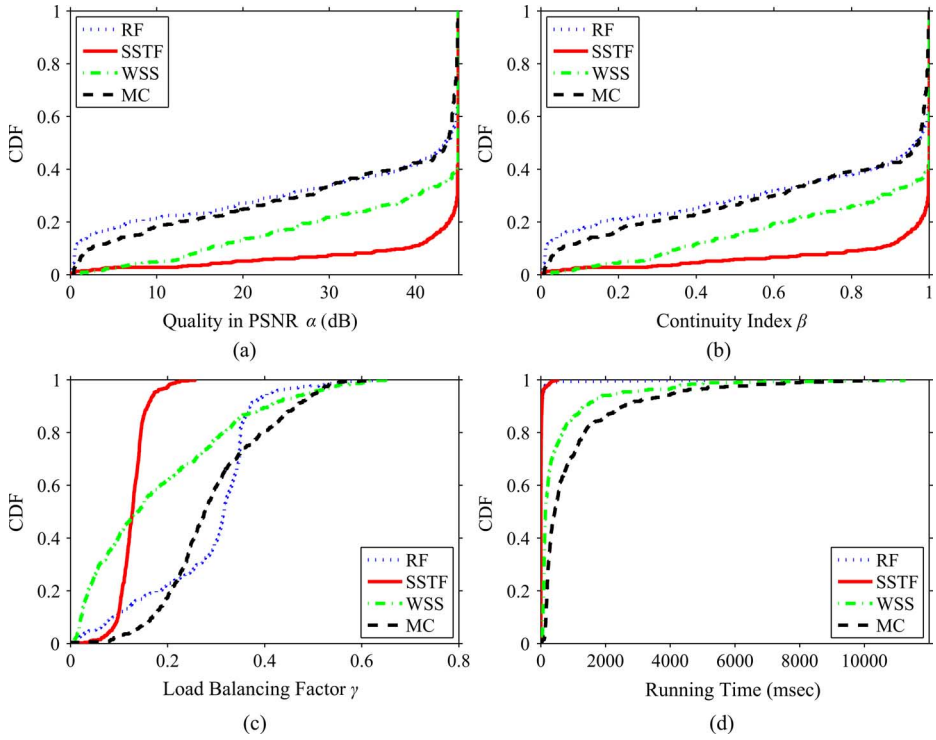


Fig. 6. Overall comparison among different algorithms. Sample results from `SonyDemo`. Note: the running time curves for the SSTF and RF algorithms are not visible in the figure because they are very close to 0. (a) Video quality. (b) Continuity index. (c) Load balance. (d) Running time.

updated for the next scheduling period. This process continues until all segments of the video have been considered by the peer (either scheduled or dropped).

4) For buffer map exchange, we use a periodic broadcasting scheme: each peer periodically checks its own buffer map (in our setup, every 5 s). If the buffer map is updated since last time checked, the peer broadcasts the updates to all its receivers. In this way, the buffer map availability information is exchanged among peers.
5) The experiment stops when all the peers have considered all segments of the video or when the experiment period expires.

The experiments involved 500 PlanetLab nodes distributed across the world. We stream the `SonyDemo` video used in the simulation. We randomly pre-deploy the video on 5% of the nodes and let other nodes join and leave the P2P network at time randomly distributed during the whole experiment period. We set the maximum number of senders to ten and use the random peer matching algorithm to find senders for each receiver. We use a scheduling window of 5 s and an initial delay of 2 s for all algorithms. The current sending rate of a sender is estimated from its previous sending rates in a moving window manner, which is a commonly used technique for bandwidth estimation. We compute the same performance metrics used in simulation study (Section VI-A) and we compute them across the successful transmissions.

D. Experimental Results

We first plot the video quality in Fig. 6(a). This figure shows that the SSTF and WSS algorithms achieve better quality than the other two. For example, at the quality level of 30 dB, there are approximately 90%, 75%, 65%, and 65% peers achieved that level, for the SSTF, WSS, RF, and MC algorithms, respectively. Next, we observe that, compared to the results from simulations, the unweighted algorithm SSTF performs better than the weighted algorithm WSS. This is because some of the nodes on PlanetLab are quite busy, as their CPU times are shared by many users. The WSS algorithm involves solving a set of linear programming problems, which usually requires considerable computation. The scheduling algorithm runs several scheduling windows ahead of current data requesting window, and if the algorithm runs slower than the data transmission speed, the clients have to wait for the scheduling results, which will degrade the performance of the algorithm. During our experiments, we observed such situation happens occasionally on some of the slow machines on PlanetLab.

We then report the continuity index in Fig. 6(b). This figure illustrates that the SSTF algorithm results in much higher continuity index than others: more that 90% of the peers can achieve a continuity index of at least 80%. The WSS algorithm comes next: more than 74% of the peers achieve the same continuity index. The RF and MC algorithms lead to poor continuity index: about 60%, at the continuity index of 80%. We next plot the load balancing factor in Fig. 6(c). This figure illustrates that both the WSS and SSTF algorithms distribute the transmission load across senders in a fairer manner than the other two algorithms. Last, we plot the average running time in Fig. 6(d). It shows that the SSTF algorithm runs faster than all the other algorithms; thus, it can be used on slow machines that have limited computational resources.

E. Summary and Remarks

We have compared the performance of the proposed WSS and SSTF algorithms against other segment scheduling algorithms in the literature. We have conducted extensive simulations using a bandwidth distribution that is commonly seen in real P2P streaming systems, where most peers are residential users and have limited bandwidth. We have also built a real prototype implementation as a proof-of-concept and deployed it on PlanetLab. We acknowledge that PlanetLab nodes are well-connected, but nevertheless, we believe that the PlanetLab deployment still sheds some lights on the performance of our algorithms in the Internet. As we mentioned in Section VII, we are currently working on evaluating the proposed algorithms in real deployments.

Last, we note that the streaming performance from the PlanetLab experiments are generally better than that in the simulations, e.g., compare the continuity index reported in Fig. 3(b) and (b). The difference is largely due to the bandwidth setup: we configure the simulations to use a realistic peer bandwidth distribution extracted from real bandwidth-constrained P2P users, while we do not limit peer bandwidth in PlanetLab experiments. Such diverse setups allow us to evaluate the proposed algorithms under wider ranges of network environments.

VII. Conclusions and Future Work

We studied the multi-sender data transmission problem in P2P video streaming systems, where a receiver periodically computes a transmission schedule for its senders to maximize its perceived video quality. We formulated the general version of the problem as an ILP formulation, where different segments have different weights in terms of video quality improvements. Optimally solving this ILP problem, however, may take prohibitively long time and is not suitable for P2P video streaming systems. Thus, we proposed the WSS algorithm that runs in polynomial time and achieves an approximation factor of 3 in the worst case. We also formulated a simplified version of the problem with another ILP formulation, where all segments have the same weight. We proposed the SSTF algorithm for the simplified model, and we showed that it has an approximation factor of 2.

To the best of our knowledge, the WSS and SSTF algorithms are the first two P2P segment transmission algorithms that provide guarantees on the worst-case performance. For the average-case performance, we evaluated our algorithms using simulation and PlanetLab experiments using several application-layer performance metrics. The used performance metrics are: 1) perceived video quality, 2) continuity index, and 3) load balance factor. Our extensive simulation and experimental results indicate that the proposed WSS and SSTF algorithms not only provide analytical guarantees on the worst-case performance, but they also have superior application-layer performance on the average case compared to other scheduling algorithms proposed in the literature and used in current P2P streaming systems. Furthermore, our algorithms are computationally efficient and thus can be implemented in actual P2P streaming systems.

The work in this paper can be extended in several directions. For example, although the PlanetLab experiments shed some light on the application-level performance of the proposed WSS and SSTF algorithms in the Internet, PlanetLab nodes may not accurately represent actual peers. Deploying and evaluating our algorithms in a widely-used P2P streaming system, such as Vuze [43], is one of our future works. The actual deployment can also be used to evaluate the applicability of the proposed algorithms in P2P live streaming systems. Another direction for future work is to develop new scheduling algorithms with smaller approximation factors.

References

[1] B. Liu, Y. Cui, B. Chang, B. Gotow, and Y. Xue, "BitTube: Case study of a web-based peer-assisted video-on-demand system," in *Proc. IEEE Int. Symp. Multimedia (ISM'08)*, Berkeley, CA, Dec. 2008, pp. 242–249.

[2] P. Rodriguez, S. Tan, and C. Gkantsidis, "On the feasibility of commercial, legal P2P content distribution," *ACM SIGCOMM Comput. Commun. Rev. (CCR'06)*, vol. 36, no. 1, pp. 75–78, Jan. 2006.

[3] Y. Tu, J. Sun, M. Hefeeda, and S. Prabhakar, "An analytical study of peer-to-peer media streaming systems," *ACM Trans. Multimedia Comput., Commun., Appl.*, vol. 1, no. 4, pp. 354–376, Nov. 2005.

[4] D. Xu, S. Kulkarni, C. Rosenberg, and H. Chai, "Analysis of a CDN-P2P hybrid architecture for cost-effective streaming media distribution," *ACM/Springer Multimedia Syst. J.*, vol. 11, no. 4, pp. 383–399, Apr. 2006.

[5] J. Liu, S. Rao, B. Li, and H. Zhang, "Opportunities and challenges of peer-to-peer Internet video broadcast," *Proc. IEEE*, vol. 96, no. 1, pp. 11–24, Jan. 2008.

[6] Y. Chu, S. Rao, and H. Zhang, "A case for end system multicast," in *Proc. ACM SIGMETRICS'00*, Santa Clara, CA, Jun. 2000, pp. 1–12.

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

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

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

[10] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. Mohr, "Chainsaw: Eliminating trees from overlay multicast," in *Proc. Int. Workshop Peer-to-Peer Systems (IPTPS'05)*, Ithaca, NY, Feb. 2005, pp. 127–140.

[11] S. Annapureddy, S. Guha, C. Gkantsidis, D. Gunawardena, and P. Rodriguez, "Is high-quality VoD feasible using P2P swarming?," in *Proc. Int. World Wide Web Conf. (WWW'07)*, Banff, AB, Canada, May 2007, pp. 903–912.

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

[13] C. Liang, Y. Guo, and Y. Liu, "Is random scheduling sufficient in P2P video streaming?," in *Proc. IEEE Int. Conf. Distributed Computing Systems (ICDCS'08)*, Beijing, China, Jun. 2008, pp. 53–60.

[14] M. Zhang, Q. Zhang, L. Sun, and S. Yang, "Understanding the power of pull-based streaming protocol: Can we do better?," *IEEE J. Select. Areas Commun.*, vol. 25, no. 9, pp. 1678–1694, Dec. 2007.

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

[16] M. Zhang, Y. Xiong, Q. Zhang, L. Sun, and S. Yang, "Optimizing the throughput of data-driven peer-to-peer streaming," *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 1, pp. 97–110, Jan. 2009.

[17] J. Chakareski and P. Frossard, "Utility-based packet scheduling in P2P mesh-based multicast," in *Proc. SPIE Int. Conf. Visual Communication and Image Processing (VCIP'09)*, San Jose, CA, Jan. 2009.

[18] PPLive. [Online]. Available: http://www.pplive.com/.

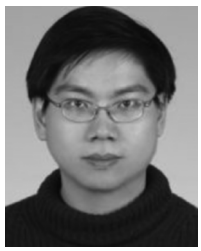
[19] X. Hei, C. Liang, J. Liang, Y. Liu, and K. Ross, "A measurement study of a large-scale P2P IPTV system," *IEEE Trans. Multimedia*, vol. 9, no. 8, pp. 405–414, Dec. 2007.

[20] G. Kowalski and M. Hefeeda, "Empirical analysis of multi-sender segment transmission algorithms in peer-to-peer streaming," in *Proc. IEEE Int. Symp. Multimedia (ISM'09)*, San Diego, CA, Dec. 2009, pp. 243–250.

[21] Y. Cai, A. Natarajan, and J. Wong, "On scheduling of peer-to-peer video services," *IEEE J. Select. Areas Commun.*, vol. 25, no. 1, pp. 140–145, Jan. 2007.
[22] H. Shojania, B. Li, and X. Wang, "Nuclei: GPU-accelerated many-core network coding," in *Proc. IEEE INFOCOM'09*, Rio de Janeiro, Brazil, Apr. 2009, pp. 459–467.
[23] J. Li and C. Yeo, "Content and overlay-aware scheduling for peer-to-peer streaming in fluctuating networks," *J. Netw. Comput. Appl.*, vol. 32, no. 4, pp. 901–912, Jul. 2009.
[24] C. Hsu and M. Hefeeda, "Quality-aware segment transmission scheduling in peer-to-peer streaming systems," in *Proc. ACM Multimedia Systems (MMSys'10)*, Phoenix, AZ, Feb. 2010, pp. 169–180.
[25] UUSee. [Online]. Available: http://www.uusee.com/.
[26] SopCast. [Online]. Available: http://www.sopcast.com/.
[27] TVAnts. [Online]. Available: http://www.tvants.com/.
[28] Y. Shen, "Efficient algorithms for multi-sender data transmission in swarm-based peer-to-peer streaming systems," M.Sc. thesis, Dept. Comput. Sci., Simon Fraser Univ., Surrey, BC, Canada, 2010.
[29] Y. Wang, J. Ostermann, and Y. Zhang, *Video Processing and Communications*. Englewood Cliffs, NJ: Prentice-Hall, 2002.
[30] ILOG CPLEX. [Online]. Available: http://www.ilog.com/products/cplex/.
[31] *Interior Point Algorithms: Theory and Analysis*, Y. Ye, Ed., 1st ed. New York: Wiley, 1997.
[32] H. Kellerer, T. Tautenhahn, and G. Woeginger, "Approximability and nonapproximability results for minimizing total flow time on a single machine," in *Proc. ACM Symp. Theory of Computing (STOC'96)*, Philadelphia, PA, May 1996, pp. 418–426.
[33] M. Goemans, "Improved approximation algorithms for scheduling with release dates," in *Proc. ACM-SIAM Symp. Discrete Algorithms (SODA'96)*, New Orleans, LA, Jan. 1997, pp. 591–598.
[34] A. Bar-Noy, S. Guha, J. Naor, and B. Schieber, "Approximating the throughput of multiple machines in real-time scheduling," *SIAM J. Comput.*, vol. 31, no. 2, pp. 331–352, 2001.
[35] M. Pinedo, *Scheduling: Theory, Algorithms and Systems*, 3rd ed. New York: Springer, 2008.
[36] R. M'Hallah and R. Bulfin, "Minimizing the weighted number of tardy jobs on parallel processors," *Eur. J. Oper. Res.*, vol. 160, no. 2, pp. 471–484, Jan. 2005.
[37] Video Traces Research Group, Arizona State Univ., 2009. [Online]. Available: http://trace.eas.asu.edu/h264/index.html.
[38] Z. Liu, Y. Shen, K. Ross, J. Panwar, and Y. Wang, "Substream trading: Towards an open P2P live streaming system," in *Proc. IEEE Int. Conf. Network Protocols (ICNP'08)*, Orlando, FL, Oct. 2008, pp. 94–103.
[39] K. Graffi, S. Kaune, K. Pussep, A. Kovacevic, and R. Steinmetz, "Load balancing for multimedia streaming in heterogeneous peer-to-peer systems," in *Proc. ACM Int. Workshop Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'08)*, Braunschweig, Germany, May 2008, pp. 99–104.
[40] PlanetLab Home Page. [Online]. Available: http://www.planet-lab.org/.
[41] C. Hsu and M. Hefeeda, "ISP-friendly peer matching without ISP collaboration," in *Proc. ACM Int. Workshop Real Overlays and Distributed Systems (ROADS'08)*, Madrid, Spain, Dec. 2008.
[42] D. Choffnes and F. Bustamante, "Taming the torrent: A practical approach to reducing cross-ISP traffic in peer-to-peer systems," in *Proc. ACM SIGCOMM'08*, Seattle, WA, Aug. 2008, pp. 363–374.
[43] Vuze (Azureus) BitTorrent Client. [Online]. Available: http://www.vuze.com/.

Yuanbin Shen received the B.Eng. degree in information security from the University of Science and Technology of China, Hefei, China, in 2004 and the M.Sc. degree in computing science from Simon Fraser University, Surrey, BC, Canada, in 2010.

His research interests include peer-to-peer networks, multimedia networks, and network security.



Cheng-Hsin Hsu (S'09–M'10) received the B.Sc. and M.Sc. degrees from National Chung-Cheng University, Minhsiung, Taiwan, in 1996 and 2000, respectively, the M.Eng. degree from the University of Maryland, College Park, in 2003, and the Ph.D. degree from Simon Fraser University, Surrey, BC, Canada, in 2009.

He is a Senior Research Scientist at Deutsche Telekom R&D Lab USA, Los Altos, CA. His research interests are in the area of multimedia networking and distributed systems.

Mohamed Hefeeda (S'01–M'04–SM'09) received the B.Sc. and M.Sc. degrees from Mansoura University, Egypt, in 1994 and 1997, respectively, and the Ph.D. degree from Purdue University, West Lafayette, IN, in 2004.

He is an Associate Professor in the School of Computing Science, Simon Fraser University, Surrey, BC, Canada, where he leads the Network Systems Lab. His research interests include multimedia networking over wired and wireless networks, peer-topeer systems, mobile multimedia, and Internet protocols.

Dr. Hefeeda won the Best Paper Award at the IEEE Innovations 2008 conference for his paper on the hardness of optimally broadcasting multiple video streams with different bitrates. In addition to publications, he and his students have developed actual systems, such as pCache, svcAuth, pCDN, and mobile TV testbed. The mobile TV testbed software developed by his group won the Best Technical Demo Award at the ACM Multimedia 2008 conference. He serves as the Preservation Editor of the *ACM Special Interest Group on Multimedia* (SIGMM) web magazine. He served as the program chair of the ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 2010) and as a program cochair of the International Conference on Multimedia and Expo (ICME 2011). In addition, he has served on many technical program committees of major conferences in his research areas, including ACM Multimedia, ACM Multimedia Systems, and the IEEE Conference on Network Protocols (ICNP). He is on the editorial boards of the *ACM Transactions on Multimedia Computing, Communications and Applications* (ACM TOMCCAP), the *Journal of Multimedia*, and the *International Journal of Advanced Media and Communication*.