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

Greg Kowalski
School of Computing Science
Simon Fraser University
Surrey, BC, Canada

Mohamed Hefeeda
School of Computing Science
Simon Fraser University
Surrey, BC, Canada

Abstract—**We study and analyze segment transmission scheduling algorithms in swarm-based peer-to-peer (P2P) streaming systems. These scheduling algorithms are responsible for coordinating the streaming of video data from multiple senders to a receiver in each streaming session. Although scheduling algorithms directly impact the user-perceived visual quality in streaming sessions, they have not been rigorously analyzed in the literature. In this paper, 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: (i) continuity index which captures the smoothness of the video playback, (ii) load balancing index which indicates how the load is spread across sending peers, and (iii) buffering delay required to ensure continuous playback. Our experimental analysis reveals the strengths and weaknesses of each scheduling algorithm, and provides insights for developing better ones in order to improve the overall performance of P2P streaming systems. Then, we propose a new scheduling algorithm called On-time Delivery of VBR streams (ODV). 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**

I. Introduction

Many peer-to-peer (P2P) streaming systems have been proposed in the literature and deployed in real life [1]. 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 [3] and PPLive [4]. Mesh-based systems also adapt better to network dynamics, lead to better perceived quality [5], and incur lower maintenance overhead. The goal of this paper 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. We focus on studying and optimizing the scheduling algorithms responsible for coordinating multiple senders in streaming sessions.

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 [6]. Despite the importance of segment scheduling algorithms, they have not been rigorously analyzed in the literature. For example, works such as [7], [8] only use simulations to evaluate the scheduling algorithms. Simulations, although useful for preliminary analysis, may abstract away many important practical details.

In this paper, we conduct extensive experimental study to evaluate various scheduling algorithms in the literature. 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: (i) continuity index which captures the smoothness of the video playback, (ii) load balancing index which indicates how the load is spread across sending peers, and (iii) 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 VBR streams (ODV). 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 variable bit rates (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.

The rest of this paper is organized as follows. We summarize the related work in Section II. In Section III, we describe our experimental setup. We analyze the current scheduling algorithms in Section IV, and we present the new algorithm in Section V. We evaluate and compare the performance of all scheduling algorithms in Section VI, and we conclude the paper in Section VII.

II. Related Work

A recent measurement study on PPLive [4] reports that users suffer from long start-up delays and playout lags, and suggests that better segment scheduling algorithms are needed [6]. 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 [3], [9], [10], resort to heuristic algorithms for segment scheduling. The authors of [9] propose to randomly schedule segment transmission. The authors of [3] 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 [10] 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. We analyze and compare our proposed algorithm against the ones in [3], [9].

The authors of [11] formulate an optimization problem to maximize the perceived quality, and they solve it using an iterative descent algorithm. The authors of [12] 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 paper.

Several other works are also related to the segment scheduling problem, but they do not directly solve it. 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 [13], and is not easy to deploy. Finally, several segment scheduling algorithms, such as [14], 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.

III. Experimental Setup

We conduct performance analysis and comparison of different segment transmission algorithms on the PlanetLab wide area testbed.

Our experiments involved approximately 70 different PlanetLab nodes, which were uniformly spread throughout North and South Americas, Europe, Asia, and Australia. We divided the PlanetLab nodes into 10 node groups and we conducted a similar set of experiments within each node group. The nodes within each group were selected to provide large geographical distances between each other 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 have implemented a prototype P2P streaming system, in which we implemented all of the four segment scheduling algorithms analyzed in this paper. Each active PlanetLab node ran a copy of the prototype system, which was capable of acting as a sender, a receiver, or both. We performed numerous tests cases to rigorously evaluate the performance of each algorithm. For every algorithm, we varied the number of senders from 2 to 6. For each chosen number of senders, we used a diverse set of five variable bit rate (VBR) video streams, which we obtained from [15]. Table I summarizes various characteristics of these video streams. As the table shows, the chosen video streams have quite heterogeneous visual content, bit rate, motion, number of frames, and file size. This is 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 paper.

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 specifies the test parameters, including: receiver node, sender nodes, video stream, scheduling algorithm, segment duration, and number of segments in a schedule. The test driving application also collects detailed statistics and information about the experiments, and it stores them in structured XML log files. The collected data about each video 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.

Once the test driving application specifies a receiver and the required parameters, that receiver is instructed to manage the streaming session. The receiver is responsible for invoking the scheduling algorithm to create a schedule for the next window of the requested video stream. The window size is the number of segments that will be considered for the current schedule. During our experiments, window sizes of 10, 20,

Video Name	Size (MB)	Frame Count	Mean Bit Rate (kbps)
South Park	26	30334	170
Alladin	200	89998	440
Starship Troopers	270	89998	600
Formula 1	190	44998	840
Soccer	500	89998	1100

TABLE I
VIDEO STREAMS USED IN THE EXPERIMENTS.

and 40 segments were used, and the segment size was set at 1 second. Once a 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. While the receiver node is receiving media segments, the bandwidth estimates for the sender nodes are updated. Once all the segments 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 occurs after 20 seconds), a new schedule is created and this process is repeated. The results captured by the receiver are sent back to the test driver after every completed schedule.

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

IV. ANALYSIS OF CURRENT ALGORITHMS

In this section, we briefly describe the key ideas of different segment scheduling algorithms in the literature and used in some deployed systems. We also *qualitatively* analyze these algorithms and highlight the strengths and weaknesses of each. In our analysis, we use example video sessions captured during our experiments on PlanetLab. We note that we present only sample examples to illustrate various aspects of each algorithm. The extensive, quantitative, analysis and comparison of all algorithms are presented in Sec. VI.

To analyze a given scheduling algorithm, we use what we call schedule trace graph. This graph, see Fig. 1 for an example, captures the impact of the scheduling algorithm on the quality of video streaming sessions. The y-axis of the graph represents the video time and the x-axis represents the data offset in the video stream. The video playout is represented by the red curve that traverses diagonally from the lower-left corner of the graph. Since we use VBR videos in our experiments, the slope of the playout curve can vary. If the videos were CBR, this curve would have been a straight line. The schedule trace graph also contains multiple short horizontal lines. These lines are actually composed of data points that represent video frames. Each video frame data point indicates the sequence of that particular video frame and the time that it arrives at the receiving peer. Since the sender peers transmit segments that are composed of multiple frames, we can see groups of frames arriving at the same time, forming the short horizontal lines. The frames delivered from different sender peers have different colors. The most important piece 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 meet their deadlines whereas all frames above the playout line miss their deadlines. Note that these schedule trace graphs do 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. We assume no buffering delay in the schedule trace graphs as we are more interested in the relative performance of the scheduling algorithms and not the absolute values of the continuity index.

We will only present a very small set of scheduling trace graphs, because of the space limitations. We have created and analyzed many such graphs for each algorithm.

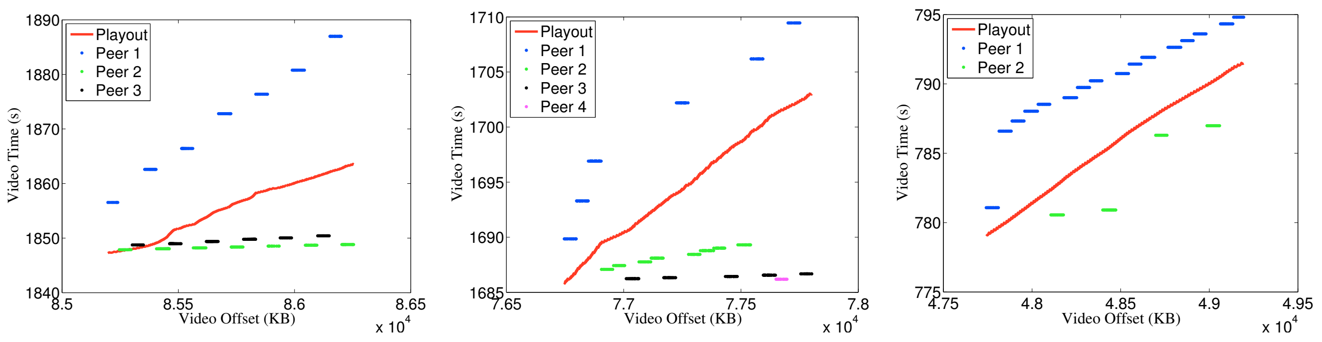
A. Round-Robin (RR) Algorithm

In this basic scheduling algorithm, every consecutive segment is assigned to the next sender in the sender list. Once we reach the end of the sender list, we start from the first sender. We continue in this manner until all segments have been assigned to a sender. 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. The round-robin (RR) algorithm, or some of its variants, is widely deployed because of its simplicity. This algorithm does not take the network and peer conditions into account, as it just equally spreads the transmission load across all peers. Thus, it will likely yield inefficient schedules, especially for heterogeneous network and peer conditions. Although simple, the RR algorithm serves as a basis for comparison against more sophisticated scheduling algorithms.

Let us now examine some streaming traces from our PlanetLab experiments. Fig. 1(a) 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 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.

B. Random Algorithm

In this scheduling algorithm, every segment is assigned to a random sender from the sender list [9]. 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 schedules. This algorithm is very simple and much like the RR algorithm, it servers as a basis for comparison against other scheduling algorithms. The main disadvantage



(a) Round Robin: failed schedule (b) Random: failed schedule (c) Rarest First: failed schedule

Fig. 1. Analysis of the round-robin, random, and rarest first algorithms.

is that there is a low probability that the resulting assignment will be anywhere close to an optimal schedule.

Let us now examine some streaming traces from PlanetLab experiments. In Fig. 1(b) we can see a segment assignment created by the random scheduler. Although it's possible for a random algorithm to come up with an efficient schedule, this is not the case here. Fig. 1(b) shows an inefficient schedule created by the random algorithm where the schedule assigns most of the early segments to the slowest sender (peer 1), resulting in all those segments missing their deadlines.

C. *Rarest First (RF) Algorithm*

The Rarest First (RF) scheduling algorithm is used in several deployed P2P streaming systems, including the popular CoolStreaming/DONet system [3]. This algorithm first calculates the number of potential suppliers for each segment (i.e., the partners containing the segment in their buffers). The assumption 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 [3].

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 or a few peers even though there may be multiple partners available. In large scale systems, this may lead to overloading of fast 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.

We show in Fig. 1(c) a trace collected from the PlanetLab experiments that used the RF algorithm. As mentioned earlier, one of the characteristics of the RF algorithm is that it tends to overutilize the faster senders and underutilize the slower ones. Relying on a few senders may not be ideal in practical settings, since communication paths among peers often dynamically change during a video session. Fig. 1(c) shows an example case that suffered from this situation. At the time of scheduling, peer 1 was the fastest sender and it got assigned most segments. At run time, the network communication got delayed and as a result many segments missed their deadlines.

V. Proposed Scheduling Algorithm (ODV)

We propose a new segment scheduling algorithm, which we call On-time Delivery of VBR streams (ODV). The goal of this algorithm is to maximize the number of segments that meet their deadlines by assigning them to peers that will deliver them earlier. As demonstrated by our extensive evaluation in Sec. VI, our algorithm leads to better user-perceived visual quality, balances the loads across peers and efficiently utilizes most of the available peers.

One of the key features of the proposed algorithm is that it explicitly considers the inherent variability in the bit rates of encoded video streams. In particular, it does not assume that video segments with the same time duration have necessarily the same byte size. This is more realistic as most encoded videos have variable bit rates because of the different compression methods used for different frame types as well as the diverse visual complexities of video frames. This is unlike most current scheduling algorithms, which estimate the size of each segment by using the average bit rate of the video stream.

To illustrate the importance of considering variable segment sizes, we analyzed several video streams. A sample of our results is shown in Fig. 2, where we plot the segment size (in KB) for each 1-second video data. The horizontal line represents the estimated segment size, which is based on the computed average bit rate for the entire video stream. The fluctuating curve represents the actual segment size. The

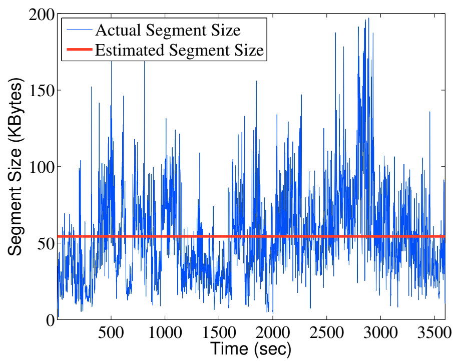


Fig. 2. Variability of segment sizes in video streams (Aladdin video).

figure clearly shows the variability in segment sizes. Notice that each segment contains all video frames (24–30) in the corresponding 1-second period of the video, which include different I, P, and B frames. The figure does not plot the sizes of individual video frames, rather the sum of all of them in 1 second. In all instances where the actual segment sizes are larger than the estimated constant size, the risk of inefficient scheduling increases. This is because we could assume that, for example, each 1-second segment is 55 KB whereas the actual 1-second segments vary between 70 and 80 KB for a particular piece of the video. So even if we have accurate bandwidth estimates, the segment assignments will likely lead to video frames missing their playback deadlines. On the other hand, if the actual segment sizes are smaller than the estimated constant size, the resources of the sender peers will be underutilized because they will be idle for some time.

A high-level pseudo code of the proposed scheduling algorithm is shown in Fig. 3. This algorithm is executed by the receiver in a streaming session. 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. 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. Our 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

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.          $p_s$ = $\varnothing$ // selected sender
5.          $t_{ed}$ = $N_{max}$ // earliest delivery time
6.          foreach $p$ in suppliers[$s$]
7.          // potential supplier peer $p$ for segment $s$
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
```

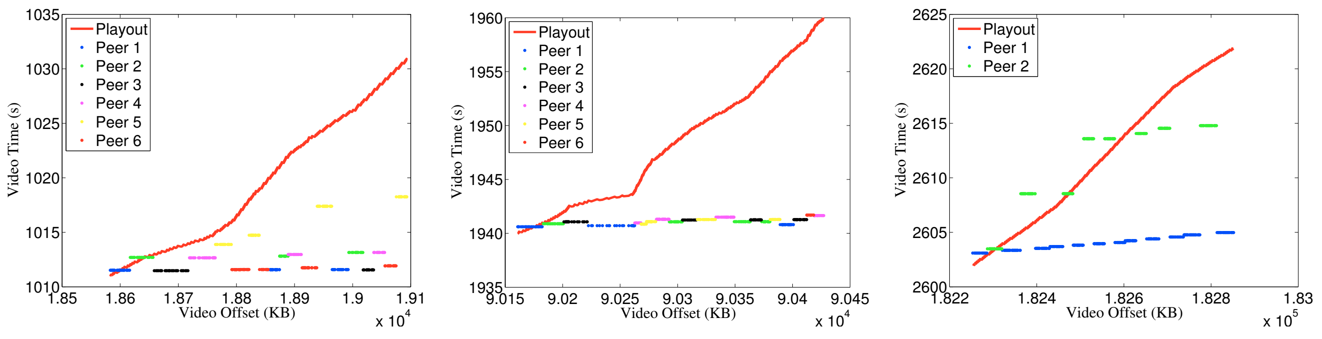
Fig. 3. The proposed segment scheduling algorithm.

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 RF algorithm. However, unlike the RF 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. Thus, the ODV algorithm may actually assign segments to slower peers before fully saturating faster peers, if the slower peers will deliver them earlier. Thus, the ODV algorithm also achieves better load sharing by utilizing the slower senders.

We now examine some streaming traces from the PlanetLab experiments to illustrate the ODV algorithm. In Figs. 4(a) and 4(b) we have two 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 RF algorithm. Fig. 4(c) 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. As a result there may be some segments that miss their deadlines. This can be handled, however, using different techniques such as buffering delay.

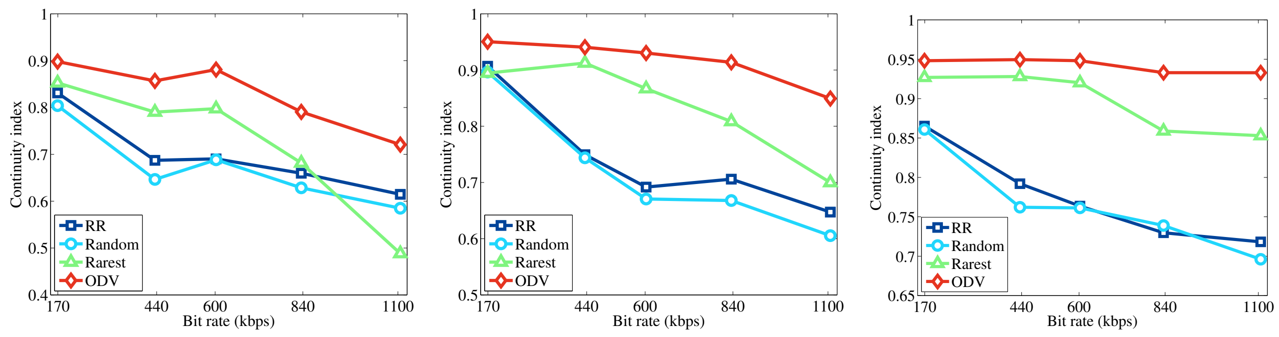
VI. Evaluation

In this section, we conduct extensive evaluation and comparison of all segment scheduling algorithms described in this



(a) Good schedule (b) Good schedule (c) Bad schedule

Fig. 4. Analysis of the proposed ODV algorithm first algorithm.



(a) 3 senders (b) 4 senders (c) 6 senders

Fig. 5. Continuity index of scheduling algorithms under different scenarios.

paper, including our proposed one. The experiments were performed on the PlanetLab testbed and the setup of our experiments is described in Sec. III. As detailed in that section, we use diverse video streams, vary the number of senders, vary the location of receivers, and consider four different scheduling algorithms. For each algorithm, we conduct more than 500 different experiments and we report the average results.

We consider three important performance metrics: continuity index, load balancing across peers, and buffering delay. The continuity index captures an important angle of the user-perceived video quality, which is the smoothness of the rendered video streams. The continuity index is computed as the number of video frames that arrive before their playback deadlines over the total number of frames. Late or dropped frames cause glitches in the video playback and may cause the scene to be frozen. In our experiments, we used a reliable transport layer (TCP) so that there are no lost frames–only late frames.

The load balancing metric, referred to as the balance index, attempts to describe the relative utilization of each sender peer in a streaming session. We propose the following equation to compute the balance index:

$$\text{Balance Index} = 1 - \sum_{i=1}^{P} \left| \frac{s_i}{s_{total}} - \frac{1}{P} \right| / bi^P_{max}, \qquad (1)$$

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^P_{max} is the maximum peer delivery ratio deviation for P senders.

Let us examine some of the terms in Eq. (1). 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. (2) that describes the term bi^P_{max}

$$bi^P_{max} = \sum_{j=1}^{P} \left| s_j - \frac{1}{P} \right|, \qquad (2)$$

where $s_1 = 1$ and $s_k = 0$ for $2 <= k <= P$.

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 Eq. (1) by the term bi^P_{max}, we are essentially normalizing this sum's value. We subtract the result from Eq. (1) 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.

The third performance metric is the buffering delay, which is the time period required to preload some of the video content in order to ensure that once playback begins it will remain uninterrupted. In our experiments, we compare the scheduling algorithms based on the amount of buffering delay that are required to achieve a continuity index of 1.0, i.e., the buffering delay required to receive all video frames before their deadlines.

Due to space limitations, we only present a small sample of our results.

Results for Continuity Index. In Fig. 5, we plot the continuity index computed for different scheduling algorithms using the considered five video steams, 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 gets worse as the bit rate of the video increases. For example, in Fig. 5(b), 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 Fig. 5 is that our proposed ODV algorithm consistently outperforms the rarest first algorithm. For example, in Fig. 5(b), 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 three subfigures in Fig. 5, 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. The final observation on Fig. 5 is that increasing the number of senders generally improves the continuity index, which is intuitive as more senders bring in more streaming capacity.

Results for Load Balancing Index. A sample of our results for the load balancing index of all algorithms is demonstrated in Fig. 6. As expected, the figure shows that the round robin and random algorithms achieve a high load balancing index, which is almost one. 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. Fig. 6 shows 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.

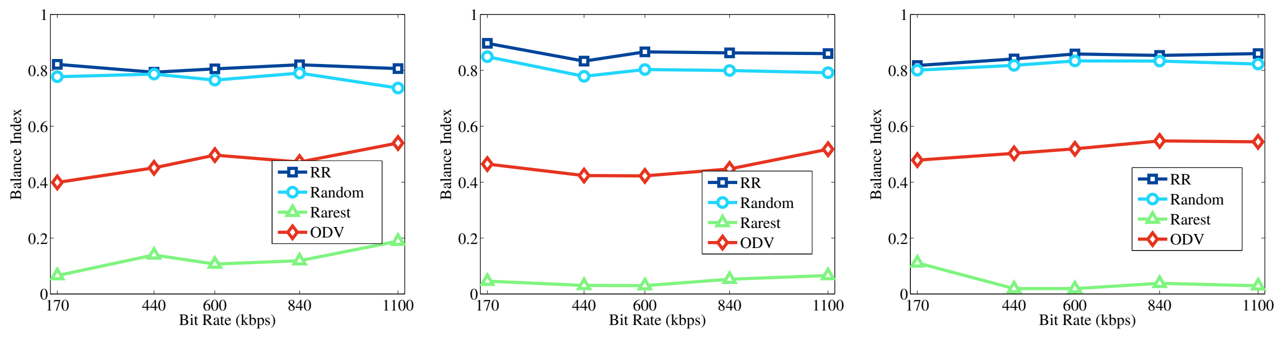
Results for Buffering Delay. Fig. 7 shows the effects of the scheduling algorithms on the buffering delay required to achieve smooth playback of the videos, again with different number of senders and for various video streams. The figure shows that the round-robin and random algorithms require substantial 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. 7(c). The results in Fig. 7 also show that our ODV algorithm always produces smaller or the same buffering delay as the rarest first algorithm.

In summary, our proposed ODV scheduling algorithm improves the playback quality by increasing the continuity index, achieves more balanced load distribution across peers, and requires small buffering delays.

VII. Conclusions

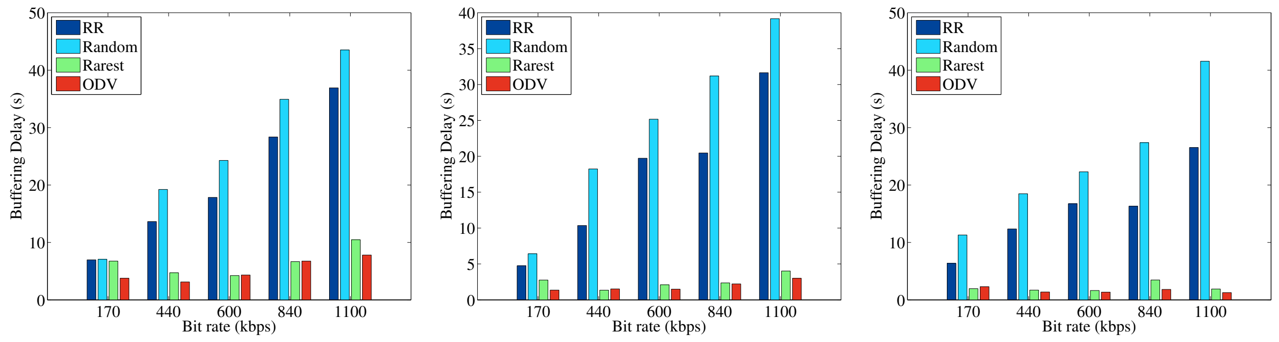
In this paper, we have experimentally analyzed three common segment transmission scheduling algorithms in P2P streaming systems: round robin, random, and rarest first. Our analysis was done by implementing a prototype P2P streaming system and deploying it on more than 70 PlanetLab nodes distributed all over the world. We conducted numerous experiments with different video streams and number of sender peers. We measured the 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 variability 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



(a) 3 senders (b) 4 senders (c) 6 senders

Fig. 6. Load balancing index of scheduling algorithms under different scenarios.



(a) 3 senders (b) 4 senders (c) 6 senders

Fig. 7. Buffering delay of scheduling algorithms under different scenarios.

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 40% of 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.

References

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

[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)*, Banff, Canada, May 2007, pp. 903–912.

[3] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum, "Coolstreaming/donet: A data-driven overlay network for peer-to-peer live media streaming," in *Proc. of IEEE INFOCOM'05*, Miami, FL, March 2005, pp. 2102–2111.

[4] "PPLive," http://www.pplive.com/.

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

[6] 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*, vol. 9, no. 8, pp. 405–414, December 2007.

[7] Y. Cai, A. Natarajan, and J. Wong, "On scheduling of peer-to-peer video services," *IEEE Journal On Selected Areas in Communications*, vol. 25, no. 1, pp. 140 – 145, 2007.

[8] 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*, vol. 25, no. 1, pp. 119 – 130, 2007.

[9] 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)*, Ithaca, NY, February 2005, pp. 127–140.

[10] 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)*, San Jose, CA, January 2005, pp. 13–25.

[11] 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, January 2009.

[12] 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*, vol. 20, no. 1, pp. 97–110, January 2009.

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

[14] J. Li and C. Yeo, "Content and overlay-aware scheduling for peer-to-peer streaming in fluctuating networks," *Journal of Network and Computer Applications*, vol. 32, no. 4, pp. 901–912, July 2009.

[15] "Video Traces," http://www.tkn.tu-berlin.de/research/trace/trace.html.