On Peer-to-Peer Media Streaming

Dongyan Xu, Mohamed Hefeeda, Susanne E. Hambrusch, Bharat Bhargava*
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907, USA
dxu@cs.purdue.edu

Abstract

Although there have been significant research efforts in peer-to-peer systems, media streaming in peer-to-peer systems has received less attention. In this paper, we study the following characteristics of a peer-to-peer media streaming system: (1) the system capacity grows dynamically; (2) peers are not supposed to exhibit server-like behavior; (3) peers are heterogeneous in bandwidth availability, and one peer may have asymmetric in-bound/out-bound bandwidth; and (4) a peer-to-peer streaming session may involve multiple supplying peers.

Based on these characteristics, we identify two new research issues and present novel solutions. The first issue is the scheduling of media data transmission from multiple peers in a peer-to-peer streaming session. Our solution is an *optimal* algorithm OTS_{p2p}. For each peer-to-peer streaming session, OTS_{p2p} computes a transmission schedule that results in the minimum buffering delay. The second issue is the fast amplification of the peer-to-peer system capacity. Our solution is a fully distributed *differentiated admission control* protocol DAC_{p2p}. By differentiating between requesting peers with different out-bound bandwidth, DAC_{p2p} (1) achieves fast system capacity amplification, (2) benefits *all* requesting peers in admission rate, waiting time, and buffering delay, and (3) creates an incentive for peers to offer their truly available out-bound bandwidth. Our simulation results show excellent performance of DAC_{p2p}.

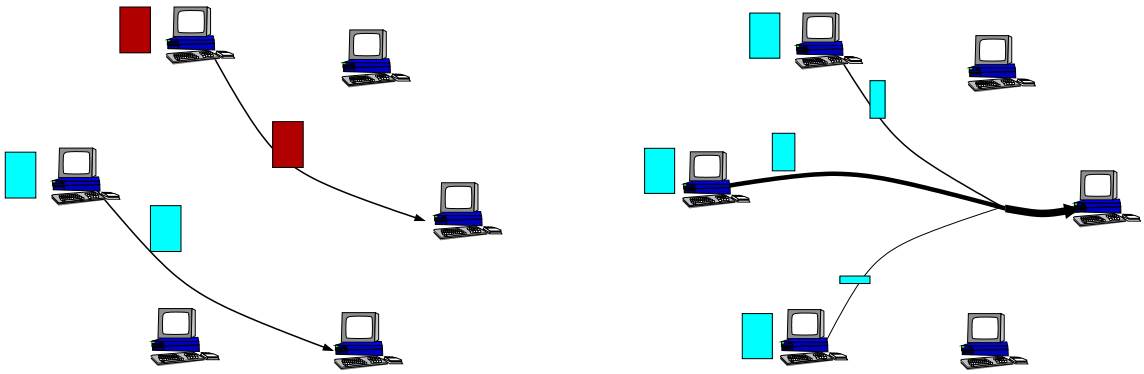
1 Introduction

'Peer-to-peer' has become an attractive distributed computing paradigm, following the popularity of (and perhaps the dispute in) current peer-to-peer applications such as Napster [3] and Gnutella [2]. In a purely peer-to-peer system, there is no centralized entity or control mechanism. Every participating peer shares the

*Submitted to IEEE International Conference on Distributed Computing Systems (ICDCS 2002), July 2-5, 2002 - Vienna, Austria. Also as Computer Science Technical Report, Purdue University, November 2001.

same responsibility as well as benefit; and the peers directly communicate with each other for data sharing and exchange, as shown in Figure 1(a).

Although there have been significant research efforts in general peer-to-peer systems during the past two years [13, 14, 17, 12, 15], one special type of peer-to-peer system has so far received much less attention: the *peer-to-peer media streaming* system. The major difference between a general peer-to-peer system and a peer-to-peer media streaming system lies in the *data sharing mode* among peers: the former uses the 'open-after-downloading' mode, while the latter uses the 'play-while-downloading' mode. More specifically, in a peer-to-peer media streaming system, a subset of peers own a certain media file, and they stream the media file to requesting peers. On the other hand, the requesting peers playback and store the media data *during* the streaming session, and they become supplying peers of the media file after the streaming session.



(a) General peer-to-peer system

(b) Peer-to-peer media streaming system

Figure 1: General peer-to-peer system and peer-to-peer media streaming system: a comparison

In this paper, we identify new research issues that arise from peer-to-peer media streaming, and present our novel solutions to these issues. We first describe the four characteristics of a peer-to-peer media streaming system that we address in this paper - the first three are shared by all peer-to-peer systems, while the last one is unique in peer-to-peer media streaming systems:

- First, a peer-to-peer media streaming system is *self-growing*. With requesting peers to become supplying peers, the system's total capacity will be amplified: the more number of peers it serves, the larger the capacity it will have.

- Second, a peer-to-peer media streaming system is *server-less*. A peer is *not* supposed to exhibit server-like behavior, such as opening a large number of simultaneous network connections. This may in practice be very harmful to other hosts sharing the same network (such as a campus network).

Therefore, the power of a peer-to-peer streaming system should lie in the large number of participating peers, rather than in the high capacity of certain individual peers.

- Third, peers in a peer-to-peer media streaming system are *heterogeneous* in their bandwidth availability and particularly, in their *out-bound* bandwidth availability. This heterogeneity may be due to the different access networks (such as Ethernet, xDSL, cable modem, and ISDN) that connect peers to the backbone; or due to the asymmetric in-bound/out-bound bandwidth of a peer. The asymmetry in turn, is either due to the access network technology (such as in the case of ADSL), or due to the asymmetric *willingness* of a peer: it is more willing to spend high in-bound bandwidth to *receive* the peer-to-peer streaming service, than to *offer* the same amount of out-bound bandwidth to *provide* the peer-to-peer streaming service.

- Finally, in a peer-to-peer media streaming session, the supplying-peer/requesting-peer relation is typically *multiple-to-one*, instead of one-to-one as in the general peer-to-peer system. This is due to (1) the real-time nature of media streaming and (2) the peers' heterogeneity in their out-bound bandwidth offers. Suppose the out-bound bandwidth offered by each supplying peer is *lower than* the original recording/playback rate of the media data[1]. In order to provide real-time streaming to the requesting peer, it is necessary to involve *multiple* supplying peers in one peer-to-peer streaming session, whose bandwidth offers add up to the original media data rate. As shown in Figure 1(b), each supplying peer will transmit a *subset* of the media data; while the requesting peer will collect and synchronize the data for real-time playback.

The above characteristics must be taken into consideration when building a peer-to-peer media streaming system. In this paper, we identify two new issues which arise from the above characteristics[2]. To the best of our knowledge, this paper presents the first in-depth study on these issues in the context of peer-to-peer media streaming:

- The first issue is the *transmission scheduling* of a peer-to-peer streaming session. More specifically, given a requesting peer and a set of supplying peers with heterogeneous (out-bound) bandwidth offers, we want to answer the following questions: For each supplying peer, how to determine the subset of media data it will transmit? For the requesting peer, how to synchronize among the supplying peers to ensure continuous playback, and how to minimize the initial media data buffering delay?

[1] We assume that a requesting peer is always willing and able to spend the amount of in-bound bandwidth equal to the original media data rate.

[2] We realize that there are other important issues in peer-to-peer media streaming, such as media data storage and location. However, they are not the focus of this paper.

- The second issue is the *fast amplification* of the peer-to-peer system capacity. For the supplying peers, their contributions to the peer-to-peer system capacity vary, due to the heterogeneity in their (out-bound) bandwidth offers. In order to quickly amplify the peer-to-peer system capacity, an intuitive approach is a *differentiated admission policy*. More specifically, among multiple requesting peers, service priority should be given to those with higher (out-bound) bandwidth offers, because they will contribute more to the peer-to-peer system capacity after becoming supplying peers. The questions are: Will fast capacity amplification ultimately benefit *all* peers? If so, What is an appropriate differentiated admission policy, and how to implement this policy in a *purely distributed* fashion?

The first issue is from the angle of an individual peer-to-peer streaming session; while the second issue is from the angle of the overall peer-to-peer streaming system. In this paper, we present our solutions to these issues. More specifically:

- For the first issue, we propose an algorithm OTS_{p2p} that computes the optimal transmission schedule for a given set of supplying peers. The optimal schedule will result in the shortest buffering delay. The requesting peer executes Algorithm OTS_{p2p} before the peer-to-peer streaming session, and then uses the computed transmission schedule to coordinate the supplying peers.

- For the second issue, we propose a distributed differentiated admission control protocol DAC_{p2p}, to be executed by both supplying and requesting peers. Compared with the current non-differentiated admission control mechanism, Protocol DAC_{p2p} achieves (1) faster amplification of peer-to-peer system capacity; (2) higher admission rate and fewer average number of rejections (before a peer is admitted) among *all* requesting peers; and (3) shorter average buffering delay among *all* admitted requesting peers. Furthermore, for (2) and (3), the protocol also differentiates between requesting peers with different (out-bound) bandwidth offers, creating an *incentive* for them to offer their truly available bandwidth.

The rest of the paper is organized as follows: we first define our peer-to-peer media streaming model in Section 2. Based on this model, Section 3 presents our first solution to the issue of transmission scheduling; while Section 4 presents our second solution to the issue of fast system capacity amplification. Section 5 evaluates the performance of the proposed protocol via extensive simulations. Section 6 compares our work with related work. Finally, Section 7 concludes this paper.

2 A Peer-to-Peer Media Streaming Model

In this section, we define a peer-to-peer media streaming model, which captures the characteristics (Section 1) of a peer-to-peer media streaming system. The model is specified in the following aspects:

(1) Roles of peers In a peer-to-peer streaming system, for each media file (such as a movie), there are *supplying peers* and *requesting peers*. Requesting peers are the ones that request the streaming of the media data. Once a requesting peer has received the peer-to-peer streaming service, it will *become* a supplying peer.

On the other hand, supplying peers are the ones that provide the media data. Since supplying peers are *not* supposed to be servers, we assume that each supplying peer can only participate in *one* peer-to-peer streaming session at any time. A majority of the supplying peers obtain the media data from the peer-to-peer streaming service they have received. However, at the beginning, there must be some 'seed' supplying peers, which obtain the media data from some external media source, such as a TV cable channel or a DVD.

(2) Storage of peers In this model, we assume that each peer in the peer-to-peer streaming system has sufficient storage to save the entire media file[3].

(3) Bandwidth of peers Let R_0 denote the original recording/playback rate of the media data. We assume in this model that each requesting peer P_r is willing and able to spend an in-bound bandwidth of $R_{in}(P_r) = R_0$, in order to receive the streaming service.

On the other hand, due to the bandwidth heterogeneity and asymmetry, the out-bound bandwidth $R_{out}(P_s)$ offered by a supplying peer P_s can only be in one of the following amounts: $\frac{R_0}{2}, \frac{R_0}{4}, \frac{R_0}{8} ... \frac{R_0}{2^N}$[4]. Therefore, in our model, one peer-to-peer streaming session will be performed by two or more supplying peers. As will be shown in Section 3, the values of $\frac{R_0}{2^n}$ $(1 \leq n \leq N)$ make it easier to solve the transmission scheduling problem.

(4) Classes of peers We then classify all peers into N classes, according to the N possible amounts of out-bound bandwidth they may offer after becoming supplying peers. More specifically, a peer willing to offer an out-bound bandwidth of $\frac{R_0}{2^n}$ $(1 \leq n \leq N)$ is called a *class-n* peer. We also assume that the *lower* the n, the *higher* the class.

(5) Capacity of the peer-to-peer streaming system We define the capacity of the peer-to-peer streaming system as the total number of peer-to-peer streaming sessions that can be provided at the same time by all

[3]The problem of heterogeneity in peers' storage is our on-going work. However, we believe that bandwidth heterogeneity is a more imperative problem than storage heterogeneity, with the signifi cant increase of disk volume in today's personal PCs.

[4]We deliberately do not assume any supplying peers with out-bound bandwidth offer of R_0, in order to highlight the *asymmetric* in-bound/out-bound bandwidth.

current supplying peers. Since a peer-to-peer streaming session involves multiple supplying peers whose $R_{out}(P_s)$ add up to R_0, the capacity of the system at time t can be computed as (suppose $\mathbf{P_s}(t)$ is the set of supplying peers in the system at t):

$$C_{sys}(t) = \lfloor \frac{\sum_{P_s \in P_s(t)}(R_{out}(P_s))}{R_0} \rfloor \quad (1)$$

(6) Segments of media data We assume that the media data can be partitioned into *small sequential segments of equal sizes*. We also assume that the media data are of Constant-Bit-Rate (CBR)[5]. Therefore, the playback time of each segment is the same, and denoted as δt. δt is typically in the magnitude of seconds.

3 Optimal Transmission Scheduling of a Peer-to-Peer Streaming Session

In this section, we study the first issue: the transmission scheduling of an individual peer-to-peer media streaming session. Based on the peer-to-peer media streaming model (Section 2), the problem can be stated as follows: For a requesting peer P_r, suppose that the set of supplying peers have been determined as $P_s^1, P_s^2, ...P_s^m$. If the following condition is true:

$$R_0 = R_{in}(P_r) = \sum_{i=1}^{m} R_{out}(P_s^i) \quad (2)$$

how to determine (1) the media data segments to be transmitted by P_s^i ($1 \leq i \leq m$); and (2) the transmission start time for P_s^i ($1 \leq i \leq m$) and the playback start time for P_r? The goal is to ensure a *continuous* playback of full media data with *minimum* buffering delay at P_r.

3.1 Impact of Transmission Schedule on Buffering Delay

Different transmission schedules may lead to different buffering delays. This is illustrated by the example in Figure 2. In this example, the set of supplying peers is $P_s^1, P_s^2, P_s^3, P_s^4$ and the requesting peer is P_r. The out-bound bandwidth offered by P_s^1, P_s^2, P_s^3, and P_s^4 are $\frac{R_0}{2}$, $\frac{R_0}{4}$, $\frac{R_0}{8}$, and $\frac{R_0}{8}$, respectively. We define the buffering delay as the time interval between the start of media data segment transmission from at least one of the supplying peers and the start of continuous playback at P_r. We also suppose that a media data segment is the *minimum buffering unit*, i.e. P_r cannot start the playback before at least one media data segment is received.

[5]The CBR media data model has been used in earlier works on media broadcasting [7]. However, we plan to study the case of Variable-Bit-Rate (VBR) media data in the future.

As shown in Figure 2, different media data assignments lead to different buffering delays. The requesting peer is P_r; and the supplying peers are $P_s^1, P_s^2, P_s^3, P_s^4$ with out-bound bandwidth of $\frac{R_0}{2}$, $\frac{R_0}{4}$, $\frac{R_0}{8}$, and $\frac{R_0}{8}$, respectively. In Assignment I, P_s^1 is assigned media data segments $8k$, $8k+1$, $8k+2$, $8k+3$ ($k = 0, 1, 2, 3...$); P_s^2 is assigned segments $8k + 4$, $8k + 5$; P_s^3 is assigned segments $8k + 6$; and P_s^4 is assigned segments $8k + 7$. The start time of playback at P_r is $5\delta t$. Therefore, the buffering delay achieved by Assignment I is $5\delta t$. However, if Assignment II is used, the buffering delay will be reduced to $4\delta t$.

3.2 An Algorithm for Computing Optimal Transmission Schedules

From the example in Figure 2, we realize that in order to achieve the minimum buffering delay, a faster supplying peer (i.e. a peer of higher class) can start its transmission *sooner* than a slower supplying peer (i.e. a peer of lower class), so that the latency of the latter can overlap with the playback of media data segments with lower sequence numbers. Furthermore, the assignment of media data segments to each supplying peer is crucial in achieving the earliest playback time, and there is *no* naive rule to determine the assignment.

We propose an algorithm OTS_{p2p}, which computes the optimal media data assignment that leads to the minimum buffering delay. The algorithm is executed by the requesting peer. After computing the media data assignment, it will initiate the peer-to-peer streaming session by notifying each participating supplying peer of the corresponding assignment. The supplying peer will then start the transmission of its assigned media data segments.

The pseudo-code of Algorithm OTS_{p2p} is shown in Figure 3. Suppose that the m supplying peers have been sorted in *descending* order according to their out-bound bandwidth offers; and that the lowest class among them is class-n. The algorithm computes the assignment of the *first* 2^n segments; and the assignment repeats itself every 2^n segments for the rest of the media file. In fact, Assignment II in Figure 2 is computed by OTS_{p2p}: after the first 'while' iteration, segments 7, 6, 5, 4 are assigned to P_s^1, P_s^2, P_s^3, and P_s^4, respectively; and P_s^3 and P_s^4 are done with the assignment. After the second 'while' iteration, segment 3 is assigned to P_s^1. During the third 'while' iteration, segments 2 and 1 are assigned to P_s^1 and P_s^2, respectively; and P_s^2 is done. During the last 'while' iteration, segment 0 is assigned to P_s^1.

The optimality of Algorithm OTS_{p2p} is stated in Theorem 1, which gives a (somewhat surprisingly) simple form of the minimum buffering delay. The proof of Theorem 1 can be found in the Appendix.

Theorem 1 *Given a set of m supplying peers $\{P_s^1, P_s^2...P_s^m\}$ and a requesting peer P_r, if we have $R_0 = R_{in}(P_r) = \sum_{i=1}^{m} R_{out}(P_s^i)$, then Algorithm OTS_{p2p} will compute an optimal media data assignment, which achieves the minimum buffering delay in the consequent peer-to-peer streaming session. The minimum buffering delay is $T_{buf}^{min} = m * \delta t$.*

3.3 Synchronizing the Supplying Peers

Algorithm OTS_{p2p} is executed by the requesting peer. After computing the optimal transmission schedule, it initiates the peer-to-peer streaming session by notifying the participating supplying peers of the transmission schedule, i.e. the corresponding (1) media data segment assignment and (2) transmission start time for each of them. We note that due to the distributed nature of peer-to-peer streaming, it is *not* possible to achieve a perfect synchronization among the supplying peers according to the transmission schedule. Fortunately, P_r can absorb the media data arrival jitter by introducing an additional buffering time upon the arrival of each segment. And this additional latency (typically in tens or hundreds of milliseconds) is trivial compared with the buffering delay (in seconds) caused by the transmission schedule.

4 Fast Capacity Amplification of a Peer-to-Peer Streaming System

In the previous section, we present our solution to the issue of transmission scheduling. However, we did not answer the question of how a requesting peer determines the set of supplying peers. In this section, we will answer this question. Moreover, we will present our solution to the second issue of fast capacity amplification of the entire peer-to-peer system. Our solution is a fully *distributed* protocol for *differentiated* admission control for requesting peers.

4.1 The Need for a Differentiated Admission Policy

Recall that one of the most salient and exciting differences between a peer-to-peer streaming system and a traditional client-server streaming system is that the capacity of the peer-to-peer system *dynamically grows*: the more number of peers served, the greater capacity the peer-to-peer system will gain. However, the issue of capacity amplification has *not* been fully addressed in the general peer-to-peer systems [13, 14, 17, 12, 15]. Furthermore, no previous work has identified the problem of capacity amplification in peer-to-peer media streaming systems with bandwidth offer heterogeneity.

To illustrate the problem in peer-to-peer system capacity amplification, we study the following simple scenario as shown in Figure 4: suppose at time t_0, there are four supplying peers in the system: two class-2 peers P_s^1 and P_s^2 (i.e. each offering out-bound bandwidth of $\frac{R_0}{4}$) and two class-1 peers P_s^3 and P_s^4 (i.e. each offering out-bound bandwidth of $\frac{R_0}{2}$). According to the system capacity definition in Section 2, the capacity of the peer-to-peer streaming system at t_0 is $\lfloor(\frac{R_0}{4} + \frac{R_0}{4} + \frac{R_0}{2} + \frac{R_0}{2})/R_0\rfloor = 1$. This means that the system can admit one requesting peer at t_0. Now, suppose there are three requesting peers: two class-2 peers P_r^1 and P_r^2 (i.e. each will offer an out-bound bandwidth of $\frac{R_0}{4}$ *after* it becomes a supplying peer); and one class-1

peer P_r^3 (i.e. will offer an out-bound bandwidth of $\frac{R_0}{2}$).

Figure 4 shows that different admission decisions lead to different degree of capacity amplification. In Figure 4(a), P_r^1 is admitted at t_0. At $t_0 + T$ (T denotes the duration of the peer-to-peer streaming session), the capacity is still 1. P_r^2 is admitted next, and at $t_0 + 2T$, the capacity becomes 2. Finally, P_r^3 is admitted. By $t_0 + 3T$, all three requesting peers are served. However, as shown in Figure 4(b), if P_r^3 is admitted at t_0, then at $t_0 + T$, the system capacity will grow to 2. Therefore, *both* P_r^1 and P_r^2 can be admitted at $t_0 + T$. By $t_0 + 2T$, all three requesting peers are served. Moreover, we define the *waiting time* of a requesting peer as the interval between its first streaming request time and the earliest time it can be admitted. Then in Figure 4(a), the average waiting time of P_r^1, P_r^2, and P_r^3 is $(0 + T + 2T)/3 = T$; while in Figure 4(b), the average waiting time is $(T + T + 0)/3 = \frac{2T}{3}$.

The simple example in Figure 4 suggests that a differentiated admission policy which favors higher-class requesting peers will lead to a faster amplification of the peer-to-peer system capacity, and will ultimately benefit requesting peers of *every* class. This is reflected by the shorter average waiting time of *all* requesting peers in the example. On the contrary, if the admission policy is non-differentiated, i.e. the admission of a requesting peer is *independent* of its future bandwidth offer after it becomes a supplying peer, the entire peer-to-peer system may suffer from slower capacity growth and longer average waiting time among *all* requesting peers. In a large scale peer-to-peer streaming system with thousands of peers, the effect of the admission policy will become more obvious, which is confirmed by our performance study in Section 5.

However, it is not easy to realize a differentiated admission policy in the peer-to-peer streaming system. Such a policy should meet the following challenges:

- First, it should lead to fast amplification of the peer-to-peer streaming system capacity. In the long term, this will benefit all requesting peers by allowing them to receive the streaming service earlier than under a non-differentiated policy. However, in the short term, the policy should *not* completely deny the admission of lower-class requesting peers.

- Second, it should be enforced in a purely distributed fashion. Due to the distributed nature of peer-to-peer systems, it is *not* desirable to assume any centralized entity which enforces the admission policy. Although centralized directory servers may exist in some peer-to-peer systems such as Napster [3], we do not assume that they will perform the admission control. Otherwise, our solution would not be applicable to other purely distributed peer-to-peer systems [2, 5, 12].

- Third, it should be differentiating in the following sense: the higher the out-bound bandwidth to be offered by a requesting peer, the greater the possibility that it will be admitted, and the shorter the

waiting time as well as buffering delay it will experience. In some sense, this differentiation creates an *incentive* to encourage requesting peers to contribute its *truly available* out-bound bandwidth to the peer-to-peer streaming system[6].

- Finally, it should be adaptive to different request/supply situations. The admission preference should be dynamically adjusted according to the current requesting/supplying condition. For example, when the arrival rate of class-1 requesting peers is low and the system has relatively sufficient capacity, the preferences to the lower-class requesting peers should be properly *elevated*.

4.2 A Distributed Admission Control Protocol

As a solution to the challenges in the previous sub-section, we present a novel distributed admission control protocol DAC_{p2p}, to be executed by every supplying peer and every requesting peer. Protocol DAC_{p2p} has two key features: (1) Each supplying peer individually decides whether or not to participate in a streaming session requested by a requesting peer. The decision is made in a probabilistic fashion, with *different* probability values applied to different classes of requesting peers. Furthermore, the probabilities are dynamically adjusted. (2) We propose a new technique called *reminder*: under certain conditions (to be detailed shortly), a requesting peer P_r may send a 'reminder' to a *busy* supplying peer P_s, reminding P_s *not to* elevate its admission preferences to requesting peers of classes lower than that of P_r.

Protocol DAC_{p2p} can be described from the angles of a supplying peer and a requesting peer:

(1) Each supplying peer P_s maintains an *admission probability vector* $< Pr[1], Pr[2]..., Pr[N] >$. $Pr[i]$ $(1 \leq i \leq N)$ will be applied to class-i requesting peers: if a class-i requesting peer contacts P_s for streaming service and P_s is *not* busy participating in another streaming session, P_s will grant the request with probability $Pr[i]$. Suppose P_s itself is a class-k peer, then the values in the probability vector of P_s is determined as follows:

- (a) Initially, when P_s becomes a supplying peer, its probability vector is initialized as follows:
  - For $1 \leq i \leq k$, we initialize $Pr[i] = 1.0$;
  - For $k < i \leq N$, we initialize $Pr[i] = \frac{1}{2^{i-k}}$.

  The intuition behind this initialization is: since P_s is a class-k peer itself, it will favor requesting peers of class-k and higher by *always* granting their streaming requests. However, for requesting peers

[6]There is, however, an important assumption here: since the bandwidth offer commitment is made when the requesting peer requests streaming service, there must be a mechanism to *enforce* the bandwidth offer commitment when the requesting peer becomes a supplying peer. It is expected that the enforcement mechanism be implemented in the peer-to-peer system software installed in each peer with proper trust and authentication. Since this is outside the scope of this paper, we simply assume that the enforcement mechanism exists.

of lower classes, it will exponentially decrease the admission probability. We call class i a *favored class* of P_s, if P_s currently has $Pr[i] = 1.0$. For example, for a class-2 supplying peer (and suppose $N = 4$), its initial admission probability vector is $< 1.0, 1.0, 0.5, 0.25 >$, and its initial favored classes are classes 1 and 2.

- (b) If P_s has been idle, then its probability vector will be updated after a timeout period of T_{out}. The update is performed as follows: for each $k < i \leq N$, $Pr[i] = Pr[i] * 2$. This means that P_s will 'elevate' the admission probabilities of lower-class requesting peers, if it has *not* been serving any requesting peer in the past period of T_{out}. If P_s remains idle, the update will be performed after every period of T_{out}, until every probability in its probability vector is 1.0, i.e. every class is P_s's favored class.

- (c) If P_s has just finished serving in a peer-to-peer streaming session, P_s will update its probability vector as follows:

  - If during the streaming session, it did not receive any request from a requesting peer of its favored class, P_s will elevate the admission probability of the lower classes, similar to the update in (b): for each $k < i \leq N$, $Pr[i] = Pr[i] * 2$.
  - If during the session, it received at least one request from a requesting peer of its favored class, the request was not granted because P_s was busy. Under a certain condition (to be described later from the angle of each requesting peer), the requesting peer left a '*reminder*' to P_s. Suppose $\hat{k}$ is the highest favored class of requesting peer(s) which left a 'reminder', then for $1 \leq i \leq \hat{k}$, $Pr[i] = 1.0$; and for $\hat{k} < i \leq N$, $Pr[i] = \frac{1}{2^{i-\hat{k}}}$.

  In the first case, P_s 'relaxes' the admission preference, because it has not been requested by any peer of its current favored classes. In the second case, P_s 'tightens' the admission preference, because there have been 'reminders' from requesting peers of its favored classes which *should have* been served, had P_s not been busy.

(2) Each requesting peer P_r first obtains a list of M randomly selected *candidate* supplying peers via some peer-to-peer service lookup/discovery mechanism[7]. We assume that the class of each candidate is also obtained. P_r then directly contacts the candidate supplying peers - from high to low classes:

- P_r will be admitted, if P_r is able to obtain permissions from enough supplying peers (among the M candidates) such that: (1) they are neither down nor busy with another streaming session; (2) they are

[7]For example, by querying a centralized directory server as in Napster [3], by using a distributed lookup service such as Chord [14], or by broadcasting the query as in [6]. However, peer-to-peer service lookup/discovery is outside the scope of this paper.

willing to provide the streaming service (i.e. having passed the probabilistic admission test); and (3) their aggregated out-bound bandwidth offer is $R_{sum} = R_0$. P_r will then execute Algorithm OTS_{p2p} to compute the transmission schedule (Section 3.2), triggers the participating supplying peers, and the peer-to-peer streaming session will begin.

- P_r will be rejected, if P_r is not able to get permission from enough supplying peers that satisfy all three conditions above. However, P_r will leave a '*reminder*' to a subset W of the *busy* candidates. W is determined as follows: from high-class to low-class busy candidates, the *first* few that satisfy the following conditions will belong to W: (1) the candidate currently favors the class of P_r; and (2) the aggregated out-bound bandwidth offer of the candidates in W is equal to $(R_0 - R_{sum})$. Each (busy) candidate in W keeps the 'reminder'; and when its current streaming session is over, it will use this reminder to update its probability vector, as described earlier from the angle of each supplying peer.

  We note that in the *reminder* technique, a reminded supplying peer may *not* in the future serve exactly the same requesting peer which left the reminder. Instead, we propose *reminder* as a distributed mechanism to realize differentiated and adaptive admission control, based on the current and overall request/supply situation in the peer-to-peer streaming system.

- If P_r is admitted, when the streaming session is over, it will become a supplying peer. If P_r is rejected, it will backoff for at least a period of T_{bkf} before making the request again. Furthermore, its backoff period will become $T_{bkf} \times E_{bkf}^{x-1}$ after the xth rejection.

As to be demonstrated in Section 5, Protocol DAC_{p2p} achieves differentiation toward different classes of requesting peers - not only in their admission probabilities, but also in the waiting time and buffering delay they experience. Moreover, the degree of differentiation is adaptive: it changes according to current request/supply situation in the system. More specifically, the 'elevate-after-timeout' technique is employed to *relax* the differentiation; while the 'reminder' technique is used to *tighten* the differentiation. Protocol DAC_{p2p} has the following configurable system parameters: M - the number of candidate supplying peers probed by a requesting peer; T_{out} - the time-out period for the elevation of lower-class admission probability; and T_{bkf} and E_{bkf} - the initial value and exponential factor to determine the back-off period for a rejected requesting peer. We will study the impact of these parameters on the protocol's performance in Section 5.

5 Performance Study

In this section, we study the performance of Protocol DAC_{p2p} using extensive simulation results. We show that Protocol DAC_{p2p} achieves the desired differentiation toward different classes of requesting peers. Furthermore, by comparing with a non-differentiated admission protocol, we show that DAC_{p2p} achieves faster system capacity amplification, as well as shorter average waiting time and buffering delay for *all* requesting peers.

5.1 Simulation Setup

We simulate a peer-to-peer media streaming system with a total of 50,100 peers. Initially, there are only 100 'seed' supplying peers, while the other 50,000 peers are requesting peers. Each 'seed' supplying peer is a class-1 peer, and it possesses a copy of a popular video file. The show time of the video is 60 minutes. The 50,000 requesting peers belong to classes 1, 2, 3, and 4, and their distribution is 10%, 10%, 40%, and 40%, respectively. This peer population distribution reflects the fact that the majority of the peer population do *not* have high out-bound bandwidth[8].

For Protocol DAC_{p2p}, its parameters are set as follows: $M = 8$ - each requesting peer probes 8 randomly selected candidate supplying peers; $T_{out} = 20min$ - each idle supplying peer elevates the admission probabilities of lower-class requesting peers every 20 minutes; and $T_{bkf} = 10min, E_{bkf} = 2$ - after the ith rejection, a requesting peer will backoff for $10 * 2^{i-1}$ minutes before retry. We will later adjust these parameters to study their impact on performance. For comparison, we also simulate a non-differentiated admission control protocol $NDAC_{p2p}$, in which the admission probability vector of each supplying peer is *always* $< 1.0, 1.0, 1.0, 1.0 >$. $NDAC_{p2p}$ also have the same values for parameters M, T_{bkf}, and E_{bkf}.

We simulate a video sharing period of 144 hours. During the first 72 hours, the 50,000 peers make their *first* streaming requests. We simulate four different arrival patterns of first-time streaming requests, as shown in Figure 5[9]. Among the request arrivals in every hour, the class distribution of the corresponding peers is the same as the overall population distribution.

5.2 Simulation Results

(1) System capacity amplification We first compare the system capacity amplification achieved by DAC_{p2p} and $NDAC_{p2p}$. Figure 6 shows the growth of the peer-to-peer system capacity with the elapse

[8] This is reported in a recent measurement study of real-world peer-to-peer systems [13].

[9] As an initial effort to understand the peer-to-peer system capacity amplification, we do not simulate the on-line/off-line state changes of the peers.

of time, under the four different (first-time) streaming request arrival patterns. Protocol DAC_{p2p} achieves significantly faster system capacity growth than $NDAC_{p2p}$, especially during the first 72 hours when the requesting peers make their first streaming requests. By the end of the 144-hour period, the system capacity achieved by DAC_{p2p} has reached at least 95% of the maximum capacity if all 50,100 peers become supplying peers. We also observe that after the first 72 hours, the system capacity growth slows down (under both protocols), because all requests are now 'retry' requests, and no new requesting peers are coming. In addition, the performance difference between DAC_{p2p} and $NDAC_{p2p}$ is the most significant under arrival pattern 3, because this bursty pattern is the most 'stressful' among the four patterns, making the need for admission differentiation more critical.

(2) Request admission rate Figure 7 shows the per-class request admission rate (accumulative over time) achieved by DAC_{p2p} and $NDAC_{p2p}$, under arrival patterns 2 and 4, respectively. We first observe that by using DAC_{p2p}, different classes of requesting peers have different admission rates (Figures 7(a) and 7(c)): the higher the class, the higher the admission rate. This admission differentiation leads to the fast system capacity amplification shown in Figure 6. On the contrary, Protocol $NDAC_{p2p}$ does *not* differentiate between classes (Figures 7(b) and 7(d)), resulting in much slower capacity amplification. Furthermore, we also observe that for requesting peers of classes 1, 2, and 3, their request admission rates in Figure 7(a) (7(c)) are *constantly higher* than those in Figure 7(b) (7(d)). Even for the class-4 requesting peers, this is also true except for the first few hours. This observation indicates that DAC_{p2p} benefits *all* classes of requesting peers.

(3) Average buffering delay Similar to (2), DAC_{p2p} also achieves both differentiation and overall benefit, in the buffering delay experienced by requesting peers of different classes. The results are shown in Figure 8. Recall that the buffering delay of a peer-to-peer streaming session is equal to δt multiplied by the number of participating supplying peers (Theorem 1 in Section 3). On the other hand, in DAC_{p2p}, if a requesting peer is admitted, it is likely that the higher the class it belongs to, the higher the classes the participating supplying peers belong to, due to the rule each supplying peer determines its favored classes. We can then infer that in DAC_{p2p}, the higher the class of an admitted requesting peer, the *fewer* the number of participating supplying peers, and therefore, the lower the buffering delay experienced by the requesting peer. Furthermore, the average buffering delay of *each* class in Figure 8(a) (8(c)) is *constantly lower* than that in Figure 8(b) (8(d)), indicating that DAC_{p2p} benefits *all* classes of requesting peers.

(4) Average waiting time Similar to (2) and (3), DAC_{p2p} also achieves both differentiation and overall benefit, in the average waiting time experienced by requesting peers of different classes. To save space, we only show in Table 1 the average (over the entire period of 144 hours) number of rejections before

admission experienced by each class of requesting peers, under the four arrival patterns. Given an average number of rejections x, the average waiting time can be computed as $T_{bkf} * E_{bkf}^{x-1}$. Again, we observe that the higher the class of admitted requesting peers, the fewer the average number of rejections each of them experiences. Furthermore, for *each class*, the average number of rejections achieved by DAC_{p2p} is fewer than that achieved by $NDAC_{p2p}$.

DAC_{p2p}/$NDAC_{p2p}$	class 1	class 2	class 3	class 4
Arrival pattern 1	1.48/3.28	1.69/3.41	2.06/3.31	2.55/3.34
Arrival pattern 2	1.77/3.73	1.93/3.75	2.40/3.72	3.15/3.74
Arrival pattern 3	2.67/4.82	3.33/4.85	3.81/4.82	4.23/4.84
Arrival pattern 4	1.93/3.45	2.19/3.46	2.59/3.42	3.16/3.46

Table 1: Per-class average number of rejections before admission (over the 144-hour period), achieved by DAC_{p2p} and $NDAC_{p2p}$

(5) Adaptivity of differentiation We now take a closer look at DAC_{p2p}'s adaptivity of admission differentiation, according to the dynamic request/supply situation in the peer-to-peer system. Recall that DAC_{p2p} uses the 'elevate-after-timeout' technique to *relax* the differentiation; while it uses the 'reminder' technique to *tighten* the differentiation. In Figure 9, we show that supplying peers use these techniques to dynamically adjust their favored classes of requesting peers, in response to the request arrival rate changes (under arrival patterns 2 and 4). The y-axis represents the lowest class of requesting peers, favored by each class of supplying peers. We observe that for each class of supplying peers, the degree of admission differentiation changes over time, roughly following the changes in the (first-time) request arrival rate. More specifically, the higher the class of supplying peers, the more sensitive they are to the changes in request arrival rate. Finally, when there are not new request arrivals, and the system capacity has grown significantly, *all* classes of supplying peers relax their admission preferences to *all* classes of requesting peers, i.e. the lowest favored class of requesting peers is 4, for all classes of supplying peers.

(6) Impact of parameters on performance Finally, we study the impact of parameters M, T_{out}, and E_{bkf} on the performance of DAC_{p2p}: Figure 10 shows the impact of M and T_{out} on the system capacity amplification; while Figure 11 shows the impact of the backoff exponential factor E_{bkf} on the request admission rate. In each study, the parameters except the one being studied remain the same as before.

In Figure 10(a), the number of candidate supplying peers probed by a requesting peer is set to 4, 8, 16, and 32, respectively. The system capacity grows significantly slower when $M = 4$, because four candidates are too few to identify sufficient number of qualified supplying peers to serve the requesting peer. If we increase M, the system capacity will grow much faster. However, when M is greater than 8, the impact of M quickly

decreases. Therefore, having a large M does not improve the system capacity growth significantly. On the other hand, it may increase the probing overhead and traffic.

In Figure 10(b), different time-out periods to relax the admission differentiation of an idle supplying peer is tried. The results indicate that T_{out} should not be too short. The explanation is: having a short time-out period may make an idle supplying peer relax its admission preferences too soon to lower-class requesting peers. Therefore, it may miss the chance to serve the ones of higher classes, when both lower-class and higher-class requesting peers are present.

In Figure 11, the backoff exponential factor E_{bkf} is set to 1, 2, 3, and 4, respectively. It is interesting to observe that exponential backoff of requesting peers does *not* help to increase the request admission rate. On the contrary, the higher the E_{bkf}, the lower the overall admission rate. In fact, the constant backoff ($E_{bkf} = 1$) scheme achieves significantly higher admission rate. Although not yet fully explored, one possible explanation is: The capacity of a peer-to-peer system is self-growing instead of fixed. Therefore, a more aggressive retry policy may actually help to increase the system capacity faster, and hence improve the overall admission rate. On the other hand, in a system with fixed capacity (such as a traditional client-server system), clients may have to perform conservative backoff, in order to achieve a high overall admission rate.

6 Related Work

Peer-to-peer file sharing systems have gained great popularity in recent years. Representative peer-to-peer systems on the Internet include Napster [3], OpenNap [4], Gnutella [2], and Freenet [5]. These systems share the same goal of de-centralized data exchange and dynamic growth of system capacity. However, they differ in their data lookup/discovery schemes. For example, Napster and OpenNap employ centralized directory servers, while Gnutella [2] uses controlled query flooding. The data sharing mode of most current peer-to-peer systems is the 'open-after-downloading' mode, not the 'play-while-downloading' (or streaming) mode as studied in this paper (however, there are exceptions, such as C-star [1] to be described later).

In the past two years, peer-to-peer systems have also attracted tremendous attention from the research community. First, there have been measurement based studies of the existing peer-to-peer systems. In [13], a detailed measurement study of Napster and Gnutella is presented, covering the bottleneck bandwidth, latency, availability, and file sharing patterns of the peers in these systems. Especially, the study reveals significant degree of heterogeneity in the peers' bandwidth availability; and it suggests that future peer-to-peer systems must have built-in incentive for peers to tell the truth about their bandwidth information. These observations have partly motivated our peer-to-peer streaming model and solutions in this paper. In

[15], based on a measurement study of OpenNap, a probabilistic model is proposed to capture the query characteristics of peer-to-peer systems with centralized directory servers.

Besides measurement studies of current peer-to-peer systems, new peer-to-peer architectures have also been proposed. These architectures focus on different aspects of a fully distributed and scalable peer-to-peer system. For example, CAN [10], Chord [14], Pastry [11], and Tapestry [17] are various distributed schemes for the lookup or location of data content in peer-to-peer systems. PAST [12] and OceanStore [8] are persistent storage services for peer-to-peer systems. Especially, they provide the management of widely replicated data, which are intrinsic in peer-to-peer systems. Our work on peer-to-peer media streaming complements these results: on one hand, we do not study the issues of peer-to-peer data lookup and storage management; on the other hand, the existing results do not address the two new issues in this paper.

Finally, several schemes of multi-source streaming have been proposed, which are perhaps more closely related to our work. In [9], a distributed video streaming framework is proposed. The scheme involves multiple replicated video senders and a single receiver. During the video streaming session, the receiver dynamically determines the sending rate of each sender; while each sender dynamically decides which packets to send. The goal is to achieve higher throughput and lower packet loss. However, their framework does not deal with the out-bound bandwidth heterogeneity among senders. Furthermore, it does not consider the issue of system capacity amplification, because the roles of sender and receiver are fixed in their system. C-star [1] is a commercial multi-source streaming service. Similar to our work, the capacity of the C-star distribution network grows over time. However, C-star does *not* differentiate between suppliers of different out-bound bandwidth capability. In addition, the available technical information about C-star does not elaborate how to determine the transmission schedule of each multi-source streaming session. In [16], an architecture called *SpreadIt* is proposed for streaming live media over a peer-to-peer network. It focuses on the dynamic construction of a multicast tree among peers requesting a live media. However, *SpreadIt* is not intended for the *asynchronous* streaming of stored media data. Also it does not deal with bandwidth heterogeneity and admission differentiation.

7 Conclusion

Peer-to-peer media streaming systems are expected to become as popular as the peer-to-peer file sharing systems. However, media streaming imposes a more stringent bandwidth requirement on participating peers. On the other hand, current peers on the Internet exhibit significant heterogeneity in their bandwidth availability. Furthermore, each peer may have (or be *willing to* offer) asymmetric in-bound and out-bound bandwidth. In this paper, we study two key issues which arise from these characteristics of peer-to-peer media streaming

systems: the first issue is the scheduling of media data transmission from multiple supplying peers involved in a peer-to-peer streaming session; while the second issue is the fast capacity amplification of the entire peer-to-peer streaming system. Our solution to the first issue is Algorithm OTS_{p2p}, which computes optimal transmission schedules for peer-to-peer streaming sessions. Our solution to the second issue is the fully distributed DAC_{p2p} protocol. By differentiating between requesting peers according their classes, DAC_{p2p} (1) achieves fast system capacity amplification, (2) benefits *all* requesting peers in admission rate, waiting time, and buffering delay, and (3) creates an incentive for peers to offer their truly available out-bound bandwidth. Our extensive simulation results demonstrate the excellent performance of DAC_{p2p}.

References

[1] C-star. *http://www.centerspan.com/*.

[2] Gnutella. *http://gnutella.wego.com*.

[3] Napster. *http://www.napster.com*.

[4] OpenNap. *http://opennap.sourceforge.net*.

[5] I. Clarke, O. Sandberg, B. Wiley, and T. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. *Proceedings of Workshop on Design Issues in Anonymous and Unobservability*, July 2000.

[6] H. Deshpande, M. Bawa, and H. Garcia-Molina. Streaming Live Media over a Peer-to-Peer Network. *Stanford Database Group Technical Report (2001-30)*, August 2001.

[7] K. Hua and S. Sheu. Skyscraper Broadcasting: A New Broadcasting Scheme for Metropolitan Video-on-Demand Systems. *Proceedings of ACM SIGCOMM '97*, September 1997.

[8] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An Architecture for Global-State Persistent Store. *Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000)*, November 2000.

[9] T. Nguyen and A. Zakhor. Distributed Video Streaming Over Internet. *Proceedings of SPIE/ACM Multimedia Computing and Networking (MMCN2002)*, January 2002.

[10] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. *Proceedings of ACM SIGCOMM 2001*, August 2001.

[11] A. Rowstron and P. Druschel. Pastry: Scalable Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems. *Proceedings of IFIP/ACM Middleware 2001*, November 2001.

[12] A. Rowstron and P. Druschel. Storage Management and Caching in PAST, a Large-Scale, Persistent Peer-to-Peer Storage Utility. *Proceedings of ACM Symposium on Operating Systems Principles (SOSP 2001)*, October 2001.

[13] S. Saroiu, P. Gummadi, and S. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. *Proceedings of SPIE/ACM Multimedia Computing and Networking (MMCN2002)*, January 2002.

[14] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. *Proceedings of ACM SIGCOMM 2001*, August 2001.

[15] B. Yang and H. Garcia-Molina. Comparing Hybrid Peer-to-Peer Systems. *Proceedings of Very Large Databases (VLDB 2001)*, September 2001.

[16] B. Yang and H. Garcia-Molina. Efficient Search in Peer-to-Peer Networks. *Stanford Database Group Technical Report (2001-30)*, October 2001.

[17] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An Infrastructure for Fault-Resilient Wide-Area Location and Routing. *UC Berkeley Computer Science Technical Report (CSD-01-1141)*, April 2001.

8 Appendix: Proof of Theorem 1

Proof. We use induction on m - the number of participating supplying peers (or simply 'peers' for the rest of the proof) in each session.

Basis For $m = 2$, according to the peer-to-peer streaming model, the only possibility is that $R_{out}(P_s^1) = R_{out}(P_s^2) = \frac{R_0}{2}$. The data assignment computed by the algorithm is: starting from time $t = 0$, P_s^1 transmits media data segments 0, 2, 4, 6, 8...; while P_s^2 transmits segments 1, 3, 5, 7... It takes each peer $2 * \delta t$ amount of time to transmit one segment. The earliest media playback time is therefore $2 * \delta t$.

Induction Suppose for $m = (k-1)$ ($k > 2$), Theorem 1 is true.

Now let $m = k$. Suppose among the k peers, the highest class is class n. We claim that there must be at least *two* class-n peers P_s^i and P_s^j among the m peers. Otherwise, if there is only one class-n peer, the sum of the other $m-1$ peers will be $R_0 * (1 - \frac{1}{2^n}) = R_0 * \frac{2^n-1}{2^n}$. On the other hand, since all other $m-1$ peers are of class $n-1$ or lower, their sum can be expressed as $R_0 * \frac{x}{2^{n-1}} = R_0 * \frac{2x}{2^n}$, where x is an integer. This leads to $2x = 2^n - 1$, a contradiction.

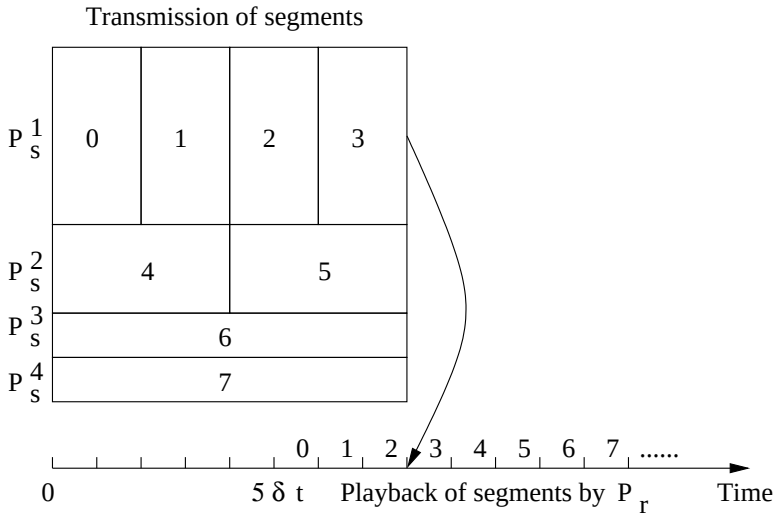
Let the two class n peers be P_s^i and P_s^j, respectively. According to the algorithm, they will be assigned segments $2^n - k$ and $2^n - k + 1$, respectively. We 'merge' P_s^i and P_s^j, and create a new class-$(n-1)$ peer P_s^u with bandwidth $\frac{R_0}{2^{n-1}}$. P_s^u also takes over segments $2^n - k$ and $2^n - k + 1$ originally assigned to P_s^i and P_s^j.

We now get a new set of $k-1$ peers. According to the induction hypothesis, the algorithm will compute the optimal data assignments to play back the following segment sequence S, with minimum buffering delay $(k-1) * \delta t$.

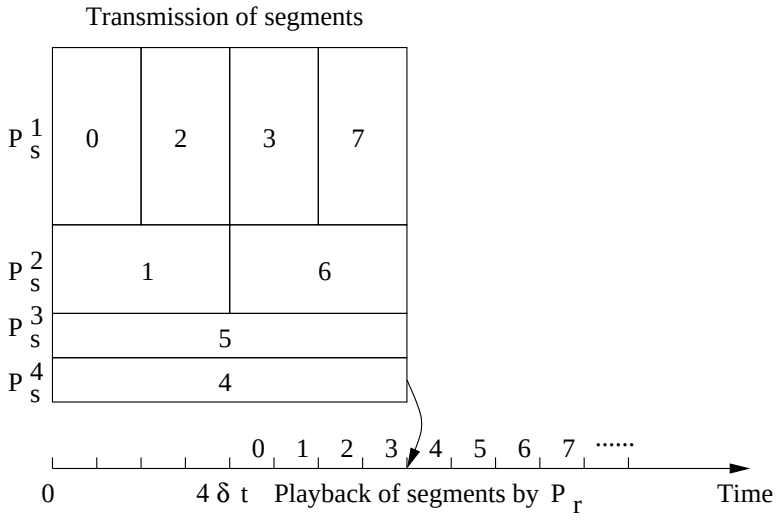
$0, 1, ...(2^{n-1} - k)$, $(\mathbf{2^n - k})$, $(2^{n-1} - k + 1), ...(2^n - k - 1), (2^n - k + 1), ...(2^n - 1)$

Sequence S is the same as sequence $0, 1, 2, ...2^n - 1$, *except* that segment $2^n - k$ is moved from its original position and placed between $2^{n-1} - k$ and $2^{n-1} - k + 1$. We now 'restore' P_s^i and P_s^j. If we make the following change to S, it will become exactly the same transmission sequence as computed by the algorithm for the k peers: segment $2^n - k$ will be transmitted by P_s^i, and will arrive at time $2^n * \delta t$, while data assignments and arrival times of other peers remaining the same as in S. This assignment leads to a minimum buffering delay of $k * \delta t$, because the last k segments $2^n - k, ...2^n - 1$ all arrive at $2^n * \delta t$. And the first $2^n - k$ segments will arrive before their respective playback time during $[0, 2^n * \delta t)$.

By now, we have shown that the statement in Theorem 1 is true for $m = k$. Therefore, Theorem 1 is true for any $m \geq 2$. **Q.E.D.**



(a) Assignment I

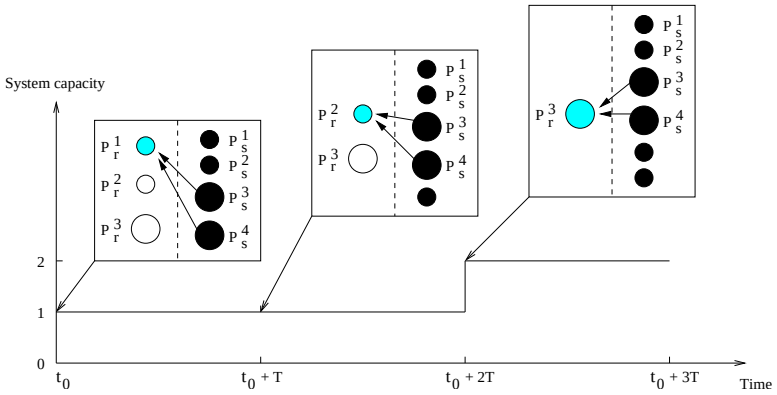


(b) Assignment II

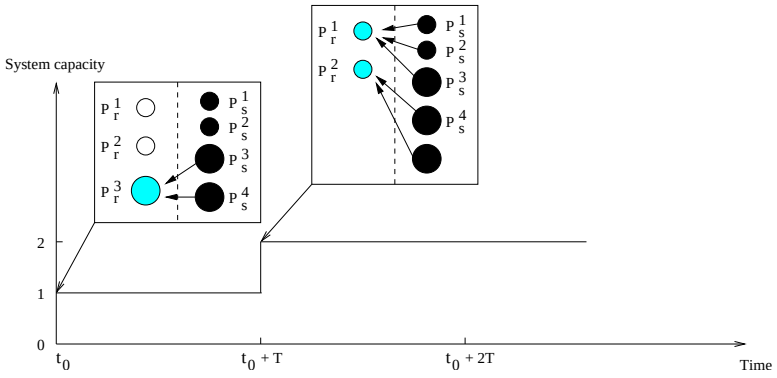
Figure 2: Two different transmission schedules for a peer-to-peer streaming session (It only shows the *first* period of data segment transmission (i.e. $k = 0$). It also shows that to ensure continuous playback, any media data segment must be completely received by P_r before the corresponding playback.)

```
OTS_p2p(P_r, {P_s^1, P_s^2 ... P_s^m}) {
    i = 2^n − 1;
    D = 2^n;
    for j = 1 to m {
        d[j] = 2^n;
        c[j] = class of P_s^j;
    }
    while (i ≥ 0) {
        for j = 1 to m
            if (d[j] = D) {
                Assign segment i to P_s^j;
                i = i − 1;
                d[j] = d[j] − 2^c[j];
            }
        D = max(d[1], d[2], ... d[m]);
    }
}
```

Figure 3: Algorithm OTS_{p2p}

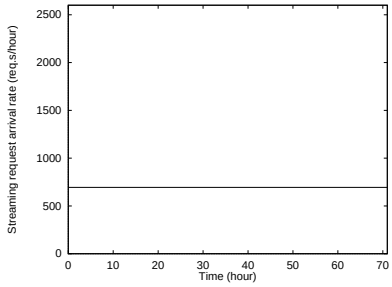


(a) Admitting $P_r^1 \rightarrow P_r^2 \rightarrow P_r^3$

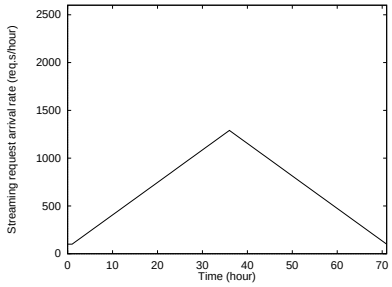


(b) Admitting $P_r^3 \rightarrow (P_r^1 + P_r^2)$

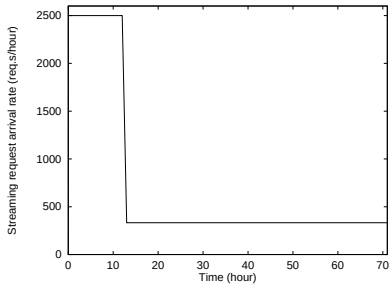
Figure 4: Different admission decisions leading to different degree of capacity amplification



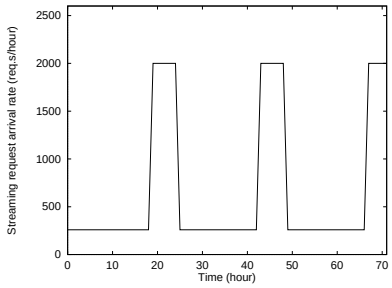
(a) Arrival Pattern 1



(b) Arrival Pattern 2

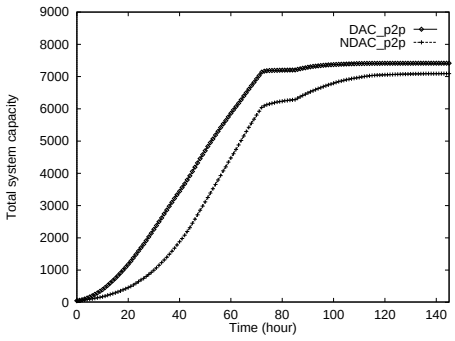


(c) Arrival Pattern 3

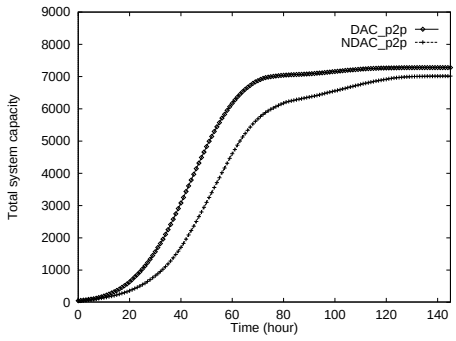


(d) Arrival Pattern 4

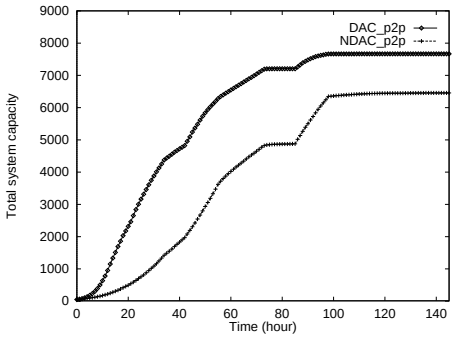
Figure 5: Four different arrival patterns of *first-time* streaming requests: Pattern 1 is '*constant* arrival rate' (in requests/hour); Pattern 2 is '*gradually increasing, then gradually decreasing* arrival rate'; Pattern 3 is '*bursty and high* arrival rate'; Pattern 4 is '*periodic bursty-and-high* arrival rate'



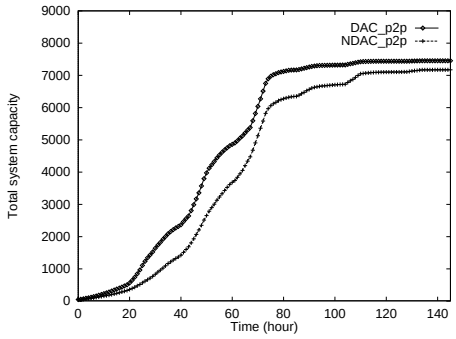
(a) Arrival pattern 1



(b) Arrival pattern 2

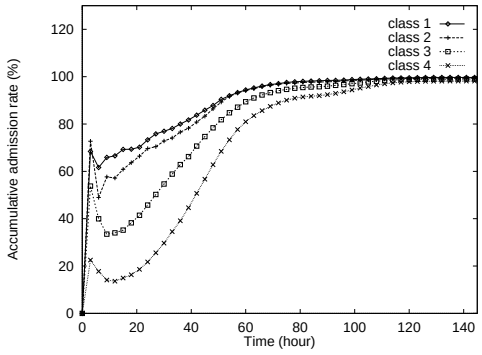


(c) Arrival pattern 3

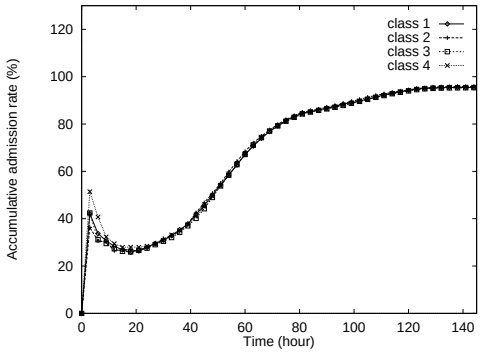


(d) Arrival pattern 4

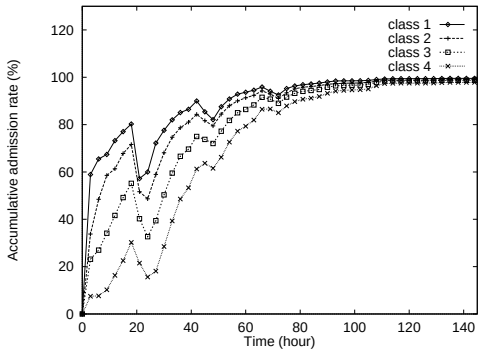
Figure 6: Capacity amplification of the peer-to-peer streaming system under different first-time request arrival patterns



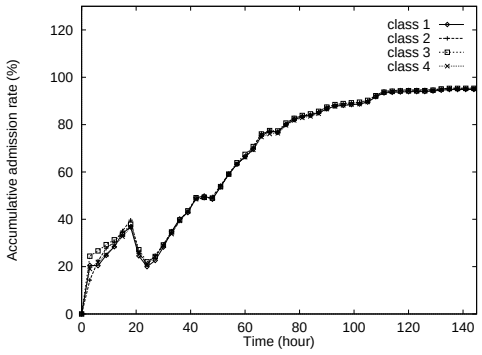
(a) Arrival pattern 2, using DAC_{p2p}



(b) Arrival pattern 2, using $NDAC_{p2p}$

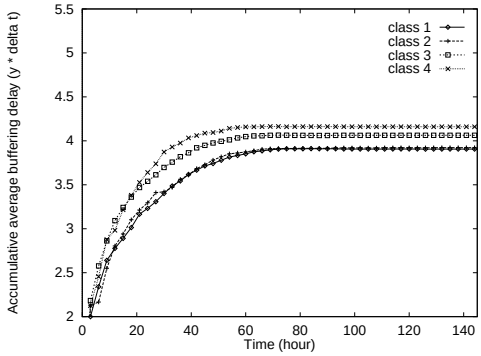


(c) Arrival pattern 4, using DAC_{p2p}

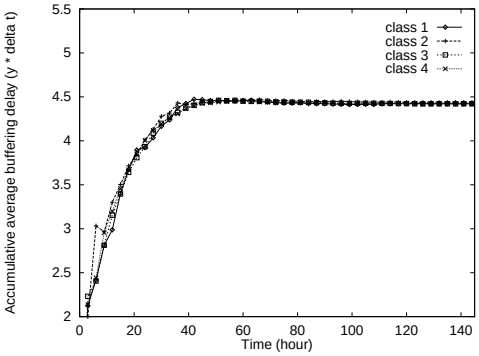


(d) Arrival pattern 4, using $NDAC_{p2p}$

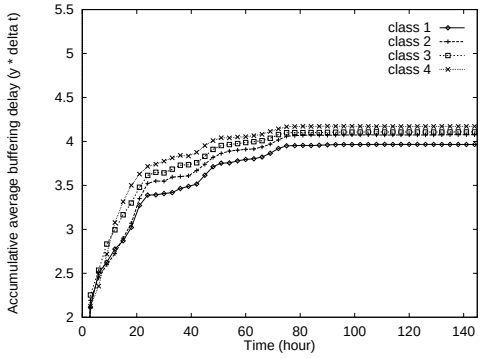
Figure 7: Per-class accumulative request admission rate achieved by DAC_{p2p} and $NDAC_{p2p}$, under arrival patterns 2 and 4



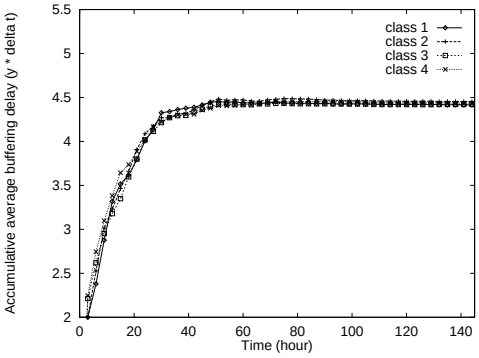
(a) Arrival pattern 2, using DAC_{p2p}



(b) Arrival pattern 2, using $NDAC_{p2p}$

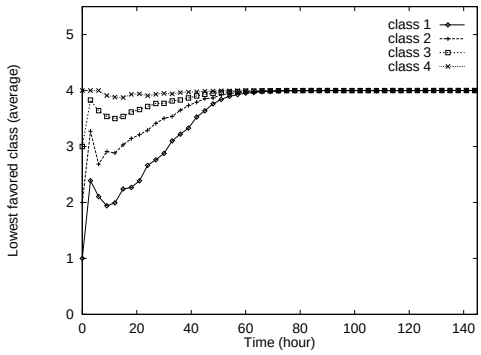


(c) Arrival pattern 4, using DAC_{p2p}

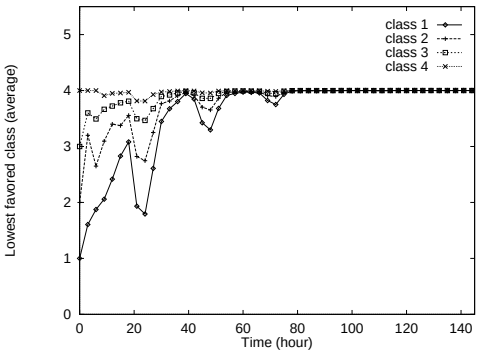


(d) Arrival pattern 4, using $NDAC_{p2p}$

Figure 8: Per-class accumulative average buffering delay achieved by DAC_{p2p} and $NDAC_{p2p}$, under arrival patterns 2 and 4 (the actual delay is $y * \delta t$)

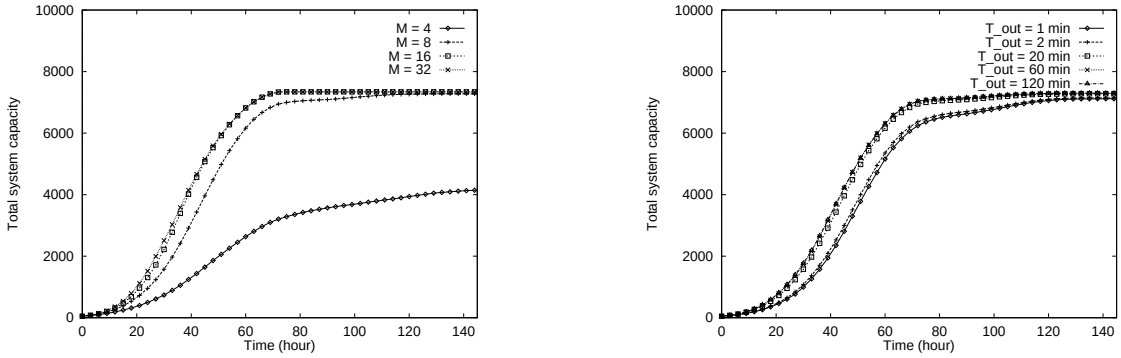


(a) Arrival pattern 2



(b) Arrival pattern 4

Figure 9: Lowest class of requesting peers, favored by each class of supplying peers (non-accumulative, averaged every 3 hours; the *higher* the y value, the *lower* the class)



(a) Impact of M

(b) Impact of T_{out}

Figure 10: Impact of M and T_{out} on system capacity amplification

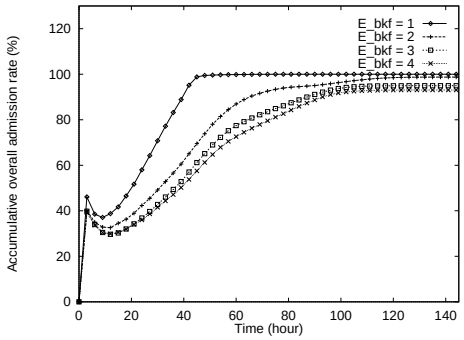


Figure 11: Impact of E_{bkf} on overall request admission rate