On Peer-to-Peer Media Streaming*

Dongyan Xu†, Mohamed Hefeeda, Susanne Hambrusch, Bharat Bhargava
Department of Computer Sciences
Purdue University, West Lafayette, IN 47907
{dxu, hefeeda, seh, bb}@cs.purdue.edu

Abstract

In this paper, we study a peer-to-peer media streaming system with the following characteristics: (1) its streaming capacity grows dynamically; (2) peers do not exhibit server-like behavior; (3) peers are heterogeneous in their bandwidth contribution; and (4) each streaming session may involve multiple supplying peers. Based on these characteristics, we investigate two problems: (1) how to assign media data to multiple supplying peers in one streaming session and (2) how to fast amplify the system's total streaming capacity. Our solution to the first problem is an optimal *media data assignment algorithm OTS_{p2p}, which results in minimum buffering delay in the consequent streaming session. Our solution to the second problem is a* distributed differentiated admission control *protocol DAC_{p2p}. By differentiating between requesting peers with different out-bound bandwidth, DAC_{p2p} achieves fast system capacity amplification; benefits* all *requesting peers in admission rate, waiting time, and buffering delay; and creates an incentive for peers to offer their truly available out-bound bandwidth.*

1. Introduction

Although there have been significant research efforts in peer-to-peer systems during the past two years [10, 11, 12, 15, 14], one category of peer-to-peer systems has so far received 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. In this paper, we assume the following four characteristics of a peer-to-peer media streaming system: the first three are shared by all peer-to-peer systems, while the last one is unique in peer-to-peer media streaming systems:

(1) A peer-to-peer media streaming system is *self-growing*. With requesting peers later becoming supplying peers, the system's total capacity will be amplified: the more peers it serves, the larger the capacity it will have.

(2) 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 connections.

(3) Peers are *heterogeneous* in their out-bound bandwidth contribution to the system. This heterogeneity may be caused either by different access networks connecting the peers, or by different willingness of the peers to contribute.

(4) The supplying-peer/requesting-peer relation is typically *many-to-one*, instead of one-to-one as in the general peer-to-peer system. Since the out-bound bandwidth offered by a supplying peer may be *less* than the original playback rate of the media data, it is necessary to involve *multiple* supplying peers in one real-time streaming session.

We identify two new problems arising in the above systems. To the best of our knowledge, this is the first in-depth study on these problems in the context of peer-to-peer media streaming. The first problem is the *media data assignment* for a multi-supplier peer-to-peer streaming session. More specifically, given a requesting peer and a set of supplying peers with heterogeneous out-bound bandwidth offers, we show how to assign a subset of the media data to each supplying peer. The second problem is the *fast amplification* of the peer-to-peer streaming capacity. Intuitively, among multiple requesting peers, service priority should be given to those who promise higher out-bound bandwidth offers, because they will contribute more to the peer-to-peer streaming capacity after becoming supplying peers.

*This work was supported in part by NSF grants CCR-9988339, CCR-0010044, CCR-001712, and CCR-001788, and CERIAS.

†Corresponding author.

We show how to realize such a differentiated admission policy, and that fast capacity amplification will ultimately benefit *all* peers.

In this paper, we propose an algorithm OTS_{p2p} that computes the optimal media data assignment for each peer-to-peer streaming session. The assignment will lead to the minimum buffering delay experienced by the requesting peer. We also 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 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 promises, 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. Sections 3 and 4 present our solutions to the two problems, respectively. Section 5 presents our simulation results. Section 6 compares our work with related work. Finally, Section 7 concludes this paper.

2 Peer-to-Peer Media Streaming Model

In this section, we define a peer-to-peer media streaming model and state our assumptions:

(1) Roles of peers For a media data item, *requesting peers* are the peers that request the data. Once the peer-to-peer streaming session is over, a requesting peer becomes a *supplying peer*. To avoid server-like behavior, each supplying peer participates in *at most one* peer-to-peer streaming session at any time. We also assume that there are some 'seed' supplying peers, which obtain the media data from some external source[1].

(2) Bandwidth of peers Let R_0 denote the playback rate of the media data. We assume that each requesting peer P_r is willing and able to set aside an in-bound bandwidth of $R_{in}(P_r) = R_0$ to receive the streaming service. However, the out-bound bandwidth $R_{out}(P_s)$ offered by a supplying peer P_s has one of the following values: $\frac{R_0}{2}$, $\frac{R_0}{4}$, $\frac{R_0}{8}$...$\frac{R_0}{2^N}$[2].

(3) Classes of peers We classify the peers into N classes, according to the N possible values of their out-bound bandwidth offer. More specifically, a peer willing to offer out-bound bandwidth $\frac{R_0}{2^n}$ $(1 \leq n \leq N)$ is called a *class-n* peer. We also assume that the *lower* the n, the *higher* the class.

(4) Capacity of the peer-to-peer streaming system We define the capacity as the total number of peer-to-peer streaming sessions that can be simultaneously provided by the system. 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 $C_{sys}(t) = \lfloor \frac{\sum_{P_s \in P_s(t)}(R_{out}(P_s))}{R_0} \rfloor$ ($\mathbf{P_s}(t)$ is the set of supplying peers in the system at t).

(5) 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 stream is of Constant-Bit-Rate (CBR) and therefore, the playback time δt of each segment is the same (δt is typically in the magnitude of seconds).

3 Optimal Media Data Assignment

In this section, we study the problem of media data assignment. Based on the model in Section 2, the problem can be stated as follows: For a requesting peer P_r and a set of supplying peers $P_s^1, P_s^2, ...P_s^m$, if $R_0 = R_{in}(P_r) = \sum_{i=1}^{m} R_{out}(P_s^i)$, determine (1) the media data segments to be transmitted by P_s^i $(1 \leq i \leq m)$ and (2) the playback start time for P_r. The goal is to ensure a *continuous* playback, with *minimum* buffering delay at P_r.

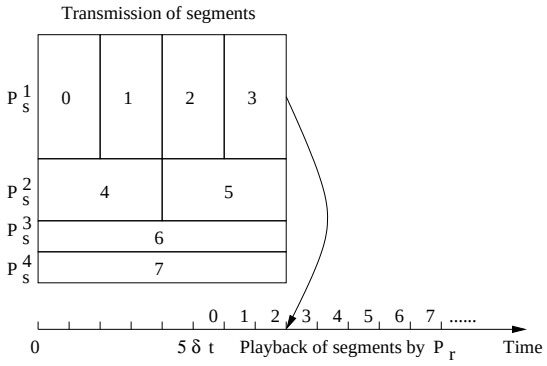
We define the buffering delay as the time interval between the start of media data segment transmission and the start of playback at P_r. As shown in Figure 1, different media data assignments lead to different buffering delays. The requesting peer is P_r; and the supplying peers are $P_s^1, P_s^2, P_s^3, P_s^4$ 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 $4\delta t$.

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.

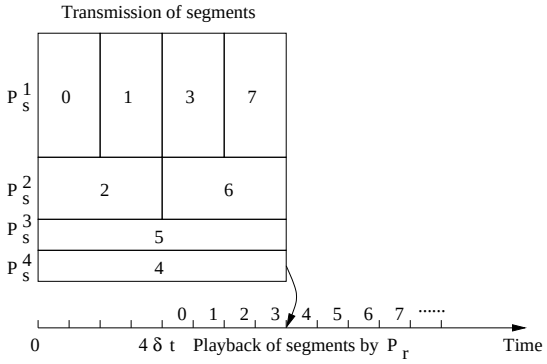
The pseudo-code of Algorithm OTS_{p2p} is shown in Figure 2. Suppose that the m supplying peers have been sorted in *descending* order according to their out-bound

[1] We also assume that each peer has suffi cient storage to store the entire media fi le.

[2] This special set of values prevents media data assignment from becoming the NP-hard binpacking-like problem.



(a) Assignment I



(b) Assignment II

Figure 1. Different media data assignments lead to different buffering delay

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 1 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, segments 3, 2 are assigned to P_s^1 and P_s^2, respectively; and P_s^2 is done. During the last two 'while' iterations, segments 1 and 0 are assigned to P_s^1 - each in one iteration.

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 [13].

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 $T_{buf}^{min} = m * \delta t$.*

$OTS_{p2p}(P_r, \{P_s^1, P_s^2 ... P_s^m\})$ {
    $i = 2^n - 1$;
    while $(i \geq 0)$ {
        for $j = 1$ **to** m
            if the assignment to P_s^j is not complete {
                Assign segment i to P_s^j;
                $i = i - 1$;
            }
    }
}

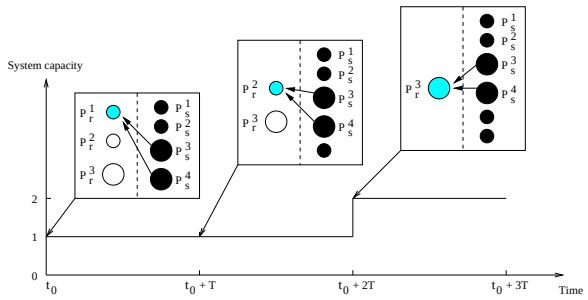
Figure 2. Algorithm OTS_{p2p}

4 Fast System Capacity Amplification

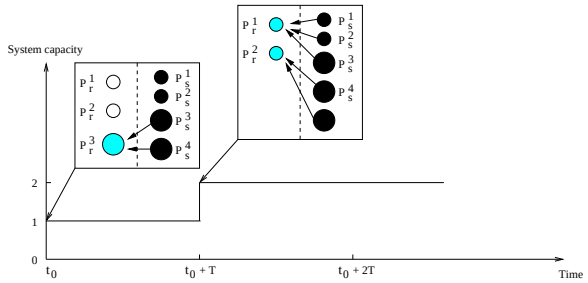
In this section, we study the problem of fast capacity amplification of the entire system. This will also answer the question of how to select a set of supplying peers for a peer-to-peer streaming session, which is not mentioned in Section 3. Recall that one of the most exciting property of a peer-to-peer streaming system is that its capacity *dynamically grows*. However, no previous work has addressed this problem in the context of peer-to-peer media streaming with peer bandwidth heterogeneity.

Consider the scenario shown in Figure 3. Suppose at time t_0, there are four supplying peers in the system: two class-2 peers P_s^1 and P_s^2 and two class-1 peers P_s^3 and P_s^4. According to the system capacity definition, 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 and one class-1 peer P_r^3. If we admit P_r^1 at t_0, the capacity will still be 1 at $t_0 + T$ (T is the duration of the peer-to-peer streaming session), and P_r^2 and P_r^3 will have to be admitted one after another at $t_0 + T$ and $t_0 + 2T$, respectively. However, if P_r^3 is admitted at t_0, the system capacity will grow to 2 at $t_0 + T$, and *both* P_r^1 and P_r^2 can be admitted at $t_0 + T$. Furthermore, we define the *waiting time* of a requesting peer as the interval between its first streaming request and the earliest time it can be admitted. The average waiting time incurred by the first admission sequence is $(0 + T + 2T)/3 = T$; while it is $(T + T + 0)/3 = \frac{2T}{3}$ in the second case.

The above example suggests that a differentiated admission policy which favors higher-class requesting peers will



(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 3. Different admission decisions lead to different growth of streaming capacity

lead to a faster amplification of the peer-to-peer system capacity, and will ultimately benefit requesting peers of *all* classes. Other requirements for such a differentiated admission policy include: (1) it should not starve the lower-class requesting peers, even in the short term; (2) it should be enforced in a purely distributed fashion; and (3) it should be differentiating such that the higher the out-bound bandwidth pledged by a requesting peer, the greater the possibility that it will be admitted, and the shorter the waiting time and buffering delay it will experience. This differentiation will create an *incentive* to encourage requesting peers to contribute its *truly available* out-bound bandwidth to the peer-to-peer streaming system[3].

Our solution is a distributed admission control protocol DAC_{p2p}. Protocol DAC_{p2p} has two key features. First, 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, and the probabilities are dynamically adjusted. Second, 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} involves operations of both supplying peers and requesting peers.

[3]There is, however, an important assumption: since the bandwidth commitment is made when a requesting peer requests streaming service, there must be a mechanism to *enforce* the bandwidth commitment after the requesting peer becomes a supplying peer. This mechanism is assumed to exist in the peer-to-peer software installed in each peer.

4.1 DAC_{p2p} - Supplying Peers

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 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 in Section 4.2), 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.

4.2 DAC_{p2p} - Requesting Peers

Each requesting peer P_r first obtains a list of M randomly selected *candidate* supplying peers via some peer-to-peer lookup mechanism[4]. 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 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 media data assignment, 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 in Section 4.1. Note that 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 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.

[4]For example, by querying a centralized directory server as in Napster [3], or by using a distributed lookup service such as Chord [12].

5 Performance Study

5.1 Simulation Setup

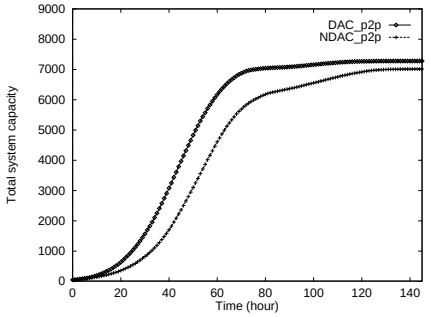
In this section, we show the excellent performance of Protocol DAC_{p2p} via extensive simulation results. We simulate a peer-to-peer media streaming system with a total of 50,100 peers. Initially, there are only 100 'seed' supplying peers, while the other 50,000 peers are requesting peers. Each 'seed' 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. Parameters in Protocol DAC_{p2p} 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. 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 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: Pattern 1 has *constant* arrivals; Pattern 2 has *gradually increasing, then gradually decreasing* arrivals; Pattern 3 has *bursty* arrivals followed by *lower and constant* arrivals; and Pattern 4 has periodic *bursty* arrivals with *low and constant* arrivals between bursts (detailed specifications are given in [13]).

5.2 Simulation Results

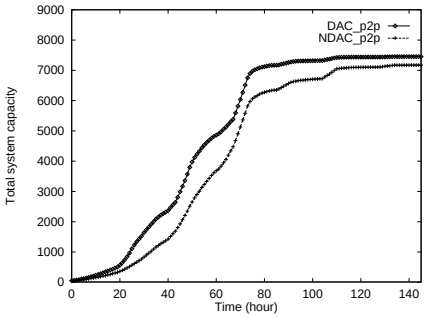
(1) System capacity amplification We first compare the system capacity amplification achieved by DAC_{p2p} and $NDAC_{p2p}$. Figure 4 shows the growth of the peer-to-peer system capacity with the elapse of time, under first-time streaming request arrival Patterns 2 and 4 [5].

[5]Results for other arrival patterns are described in [13].

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.



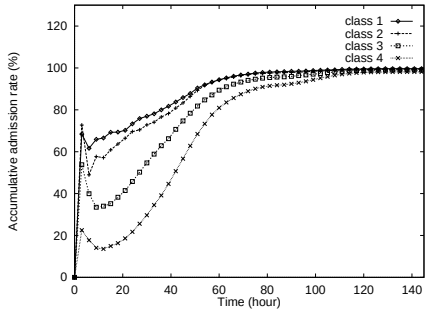
(a) Arrival Pattern 2



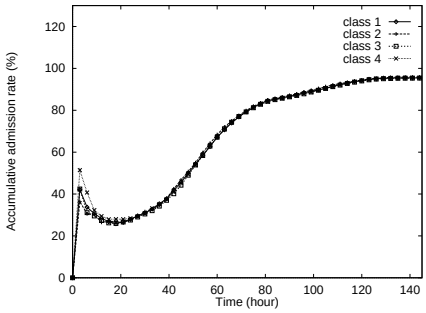
(b) Arrival Pattern 4

Figure 4. System capacity amplification using DAC_{p2p} and $NDAC_{p2p}$

(2) Request admission rate Figure 5 shows the per-class request admission rate (accumulative over time) achieved by DAC_{p2p} and $NDAC_{p2p}$, under arrival pattern 2. We first observe that by using DAC_{p2p}, different classes of requesting peers have different admission rates (Figure 5(a)): the higher the class, the higher the admission rate. On the contrary, Protocol $NDAC_{p2p}$ does *not* differentiate (Figure 5(b)), resulting in similar admission rate among all classes. Furthermore, we observe that for requesting peers of classes 1, 2, and 3, their request admission rates in Figure 5(a) are *constantly higher* than those in Figure 5(b). 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 with respect to admission rate.



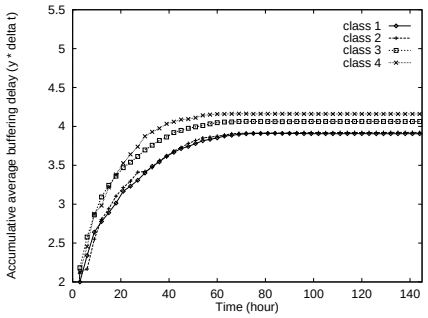
(a) Arrival Pattern 2, using DAC_{p2p}



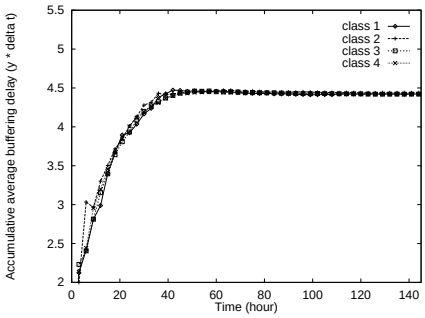
(b) Arrival Pattern 2, using $NDAC_{p2p}$

Figure 5. Per-class accumulative request admission rate

(3) Average buffering delay Similar to (2), DAC_{p2p} also achieves both differentiation and overall improvement, in the aspect of buffering delay experienced by requesting peers of different classes. The results are shown in Figure 6. 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). 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 6(a) is *constantly lower* than that in Figure 6(b).



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



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

Figure 6. Per-class accumulative average buffering delay (the actual delay is $y * \delta t$)

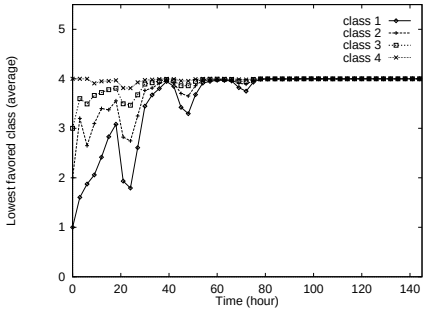
(4) Average waiting time Similar to (2) and (3), DAC_{p2p} also achieves both differentiation and overall improvement, in the aspect of waiting time experienced by requesting peers of different classes. Table 1 shows the average (over the entire period of 144 hours) number of rejections before admission experienced by each class of requesting peers, under arrival Patterns 2 and 4. 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 $NDAC_{p2p}$.

(5) Adaptivity of differentiation We now take a closer look at DAC_{p2p}'s adaptivity of admission differentiation, based on 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 7, we show that supplying peers use these techniques

Avg. rejections	Pattern 2	Pattern 4
Class 1	1.77/3.73	1.93/3.45
Class 2	1.93/3.75	2.19/3.46
Class 3	2.40/3.72	2.59/3.42
Class 4	3.15/3.74	3.16/3.46

Table 1. Per-class average number of rejections before admission ('DAC_{p2p} / $NDAC_{p2p}$')

to dynamically adjust their favored classes of requesting peers, in response to the request arrival rate changes (under arrival Pattern 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 (recall that Pattern 4 has periodic bursty arrivals). 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.



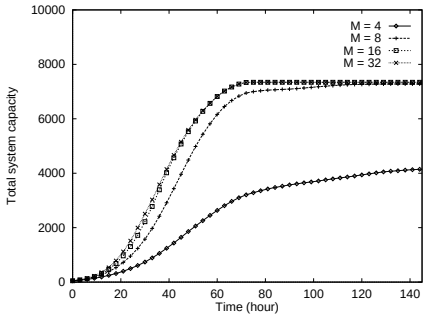
(a) Arrival pattern 4

Figure 7. Lowest class of requesting peers, favored by each class of supplying peers (non-accumulative, averaged every 3 hours

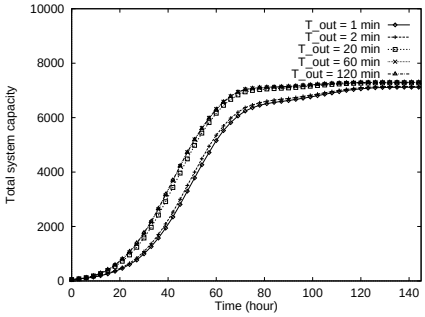
(6) Impact of protocol parameters on performance Finally, we study the impact of parameters M, T_{out}, and E_{bkf} on the performance of DAC_{p2p}: Figure 8 shows the impact of M and T_{out} on the system capacity amplification; while Figure 9 shows the impact of the backoff exponential factor E_{bkf} on the request admission rate, all under arrival Pattern 2. In each study, the parameters except the one being studied remain the same as before.

In Figure 8(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 8(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.



(a) Impact of M



(b) Impact of T_{out}

Figure 8. Impact of M and T_{out} on system capacity amplification

In Figure 9, 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.

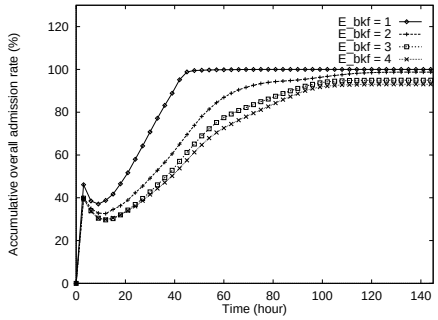


Figure 9. Impact of E_{bkf} on overall request admission rate

In summary, 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 the current request/supply situation.

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], Gnutella [2], and Freenet [4]. 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 employs 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 [11], a detailed measurement study of Napster and Gnutella is presented. 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. 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 [8], Chord [12], and Pastry [9] are distributed peer-to-peer lookup services, while PAST [10] and OceanStore [6] are peer-to-peer persistent storage services. Our work on peer-to-peer media streaming complements these results: on one hand, we do not study the problems of peer-to-peer data lookup and storage management; on the other hand, the existing results do not address the two new problems in this paper.

Finally, several schemes of multi-source media streaming have been proposed. In [7], a distributed video streaming system is presented, where each session involves multiple replicated video servers. However, it does not consider the problem of system capacity amplification, because it is still a client-server system instead of a peer-to-peer 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 [5], 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. In this paper, we study two key problems arising from peer-to-peer media streaming: the assignment of media data to multiple supplying peers involved in a peer-to-peer streaming session; and fast capacity amplification of the entire peer-to-peer streaming system. Our solution to the first problem is Algorithm OTS_{p2p}, which computes optimal media data assignments for peer-to-peer streaming sessions. Our solution to the second problem 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] 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.

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

[6] 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 ASPLOS 2000*, Nov. 2000.

[7] T. Nguyen and A. Zakhor. Distributed Video Streaming Over Internet. *Proceedings of SPIE/ACM MMCN 2002*, Jan. 2002.

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

[9] 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*, Nov. 2001.

[10] A. Rowstron and P. Druschel. Storage Management and Caching in PAST, a Large-Scale, Persistent Peer-to-Peer Storage Utility. *Proceedings of ACM SOSP 2001*, Oct. 2001.

[11] S. Saroiu, P. Gummadi, and S. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. *Proceedings of SPIE/ACM MMCN 2002*, Jan. 2002.

[12] 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*, Aug. 2001.

[13] D. Xu, M. Hefeeda, S. Hambrusch, and B. Bhargava. On Peer-to-Peer Media Streaming. *Purdue Computer Science Technical Report*, Apr. 2002.

[14] B. Yang and H. Garcia-Molina. Comparing Hybrid Peer-to-Peer Systems. *Proceedings of VLDB 2001*, Sept. 2001.

[15] 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)*, Apr. 2001.