dPAM: A Distributed Prefetching Protocol for Scalable Asynchronous Multicast in P2P Systems †

ABHISHEK SHARMA* AZER BESTAVROS IBRAHIM MATTA
{abhishek, best, matta}@bu.edu

Computer Science Department
*Electrical and Computer Engineering Department
Boston University
Boston, MA 02215, USA

***Abstract*— We leverage the buffering capabilities of end-systems to achieve scalable, asynchronous delivery of streams in a peer-to-peer environment. Unlike existing cache-and-relay schemes, we propose a distributed *prefetching* protocol where peers prefetch and store portions of the streaming media ahead of their playout time, thus not only turning themselves to possible sources for other peers but their prefetched data can allow them to overcome the departure of their source-peer. This stands in sharp contrast to existing cache-and-relay schemes where the departure of the source-peer forces its peer children to go the original server, thus disrupting their service and increasing server and network load. Through mathematical analysis and simulations, we show the effectiveness of maintaining such asynchronous multicasts from several source-peers to other children peers, and the efficacy of prefetching in the face of peer departures. We confirm the scalability of our dPAM protocol as it is shown to significantly reduce server load.**

***Index Terms*— Streaming content delivery; Peer-to-Peer systems; Asynchronous multicast; Modeling, analysis, and performance evaluation.**

I. INTRODUCTION

Motivating Application: In a large-scale peer-to-peer (P2P) network community, any node in the system may become the source of an interesting live, real-time feed that is (or may quickly become) of interest to a large number of nodes. For example, a P2P node may witness an interesting phenomenon or event, e.g., capturing a video feed from a webcam in a disaster scene, or capturing an interesting clip from a live event—say a super bowl entertainment "mishap." In such a setting, it is unrealistic to assume that all requests for such a feed will arrive synchronously. Rather, it is likely that such requests will be distributed over time, with each node interested in receiving the entire feed (or a prefix thereof). Clearly, directing all such requests for the content to the "source" of the feed (which we would term as the "server" of the content) is neither scalable nor practical. Also, using asynchronous multicast approaches requiring multicast capabilities (e.g., periodic broadcasts [26], [12], [14]) is not practical. For starters, the server may not even realize that the feed it is sharing with its P2P community is popular enough that it is "worth" multicasting! Finally, assuming that every node in the system is capable (or willing) to store the entire feed for future access by other nodes is not warranted since nodes may have limited storage capacities and/or nodes may opt to arbitrarily leave the P2P system. Rather, what is needed is a scalable protocol for the unicast dissemination of such content so that it is available on-demand to any P2P node requesting it, with minimal assumptions about the resources made available to such a protocol by the constituent nodes in the system. Specifically, it is prudent to assume that a node in the system is willing to contribute its limited storage and communication capacity as long as it is interested in receiving the content, *but not beyond*.

Leveraging Local Storage for Scalable Asynchronous Multicast in P2P Systems: Recently Jin and Bestavros proposed a scalable "cache-and-relay" approach [17] that could be used for scenarios similar to the one motivated above. Using this approach, a recipient of the feed would "cache" the most recently played out portion of the feed (after playing it out). Such cached content could then be used by other nodes in the system who request the feed within some bounded delay. This process of caching and relaying the content was shown to scale well in terms of server as well as network loads. In [6], a detailed analysis of this approach was presented.

There are two problematic aspects of the cache-and-relay approach. First, when a node leaves the system, any other nodes receiving the feed from that node are disconnected. This means that such nodes will experience a disruption in service. Second, to resume, such disconnected nodes must be treated as new arrivals, which in turn presents added load to the server (and network). This latter issue is especially significant because recent results by Jin and Bestavros [16] have shown that asynchronous multicast techniques do not scale as advertised when content is not accessed from beginning to end (e.g., due to nodes prematurely leaving the multicast and/or when non-sequential access is allowed to support VCR functionality). Specifically, Jin and Bestavros showed that techniques that ensured asymptotic logarithmic server scalability under a sequential access model would in effect behave polynomially under non-sequential access models (e.g., in the presence of premature departures and/or jumps).

† This work was supported in part by NSF grants ANI-0095988, ANI-9986397, EIA-0202067 and ITR ANI-0205294.

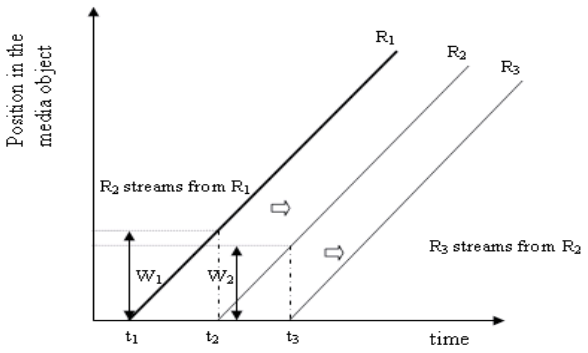

Fig. 1. Overlay-based asynchronous streaming: Illustrative Example.

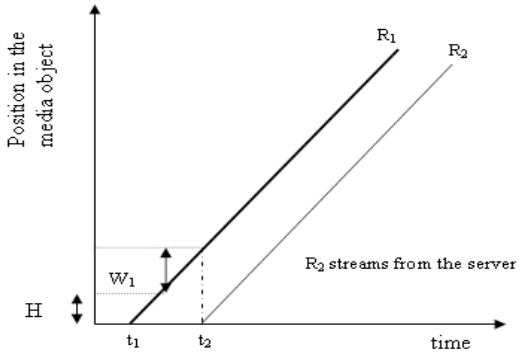

Fig. 2. Overlay-based aynchronous streaming: Illustrative Example.

Contributions: In this paper we show that a more effective use of the local storage at P2P nodes (for purposes of asynchronous multicast) must involve prefetching. In particular, rather than caching content already played out, a node is also allowed to cache content that will be played out in the future. Such prefetching is possible if we assume that a node can retrieve content at a rate higher (even if by a small factor) than the playout rate. Such an assumption is common (and realistic) [26], [12], [14]. This "lookahead" buffering capability provides each node with an opportunity to recover from the premature departures of its source. Not only does this allow the node to avoid a disruption of its playout, but also it allows the node to resume the reception of the feed from sources other than the server—thus reducing the load on the server and improving scalability. In this paper, in addition to presenting the algorithmic underpinnings of our protocol, we also provide a complete model and analysis of its scalability under a workload with independent arrivals and departures. Our analysis is backed up by extensive simulations, which demonstrate the superiority of dPAM when compared to the cache-and-relay approach studied in [17], [6].

II. Pre-fetch-and-relay: Details

In this section, we present the *Pre-fetch-and-relay* strategy for asynchronous delivery of streams through overlay networks. We illustrate the *Pre-fetch-and-relay* strategy in Figures 1 and 2. Assume that each client is able to buffer the streamed content for a certain amount of time after playback by overwriting its buffer in a circular manner. As shown in Fig.1, R_1 has enough buffer to store content for time length W_1; i.e. the data cached in the buffer is replaced by fresh data after an interval of W_1 time units. When the request R_2 arrives at time $t = t_2$, the content that R_2 wants to download is available in R_1's buffer and, hence, R_2 starts streaming from R_1 instead of going to the server. Similarly, R_3 streams from R_2 instead of the server. Thus, in Fig.1, leveraging the caches at end-hosts helps to serve three clients using just one stream from the server.

In Fig.2, by the time request R_2 arrives, part of the content that it wants to download is missing from R_1's buffer. This missing content is shown as H in Fig.2. If the download rate is the same as the playout rate, then R_2 has no option but to download from the server. However, if the network (total) download rate is greater than the playback rate, then R_2 can open two simultaneous streams - one from R_1 and the other from the server. It can start downloading from R_1 at the playback rate (assuming that R_1's buffer is being overwritten at the playback rate [1]) and obtain the content H from the server. After it has finished downloading H from the server, it can terminate its stream from the server and continue downloading from R_1. This stream patching technique to reduce server bandwidth was proposed in [13] . Assuming a total download rate of α bytes/second and a playback rate of 1 byte/second, the download rate of the stream from the server should be $\alpha - 1$ bytes/second. Hence, for this technique to work $\alpha - 1 \geq 1 \Rightarrow \alpha \geq 2$. Hence, we need the total download rate to be at least twice the playback rate for stream patching to work for a new arrival.

In the event that a client departs from the peer-to-peer network, all clients downloading from the buffer of the departing client will have to switch their streaming session either to some other client, or the server. The stream patching technique can be used by a client in this situation as well to avoid downloading from the server. As shown in Section II-C, unlike the case for a new arrival, the stream patching technique may work in this situation even when the total download rate is less than twice the playout rate, i.e. $\alpha < 2$.

When the download rate is greater than the playout rate, a client can *pre-fetch* content to its buffer before it is time to playout that content. Pre-fetching content can help achieve a better playout quality in overlay multicast. In a realistic setting, there would be a certain delay involved in searching for a peer to download from; for example, consider the situation depicted in Fig. 3. R_3 starts streaming from R_2 on arrival. After R_2 departs, as shown in Fig. 3, it takes R_3 **D** seconds (time units) to discover the new source of download R_1. If the pre-fetched "future" content in R_3's buffer, at the time of R_2's departure, requires more than **D** seconds (time units) to playout (i.e. the size of the future content is greater than **D** bytes, assuming

[1]We discuss the condition under which R_1's buffer will be refreshed at the playout rate instead of the download rate in Section III.

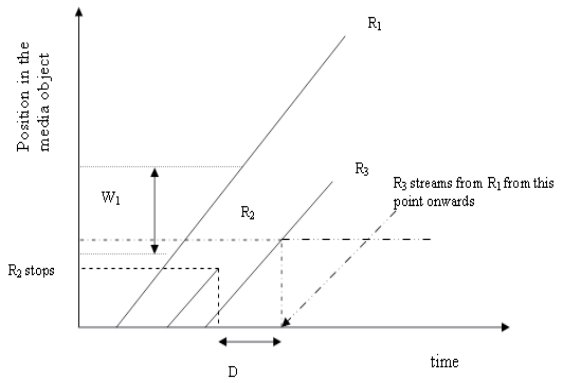

Fig. 3. Delay in finding a new source for a download: Illustration.

a playout rate of 1 byte/second) then the playout at R_3 does not suffer any disruption on R_2's departure. If the size of the "future" content is smaller than **D** bytes, then R_3 will have to open a stream from the server, after it has finished playing out its pre-fetched content, till it discovers R_1. In a *Cache-and-relay* strategy, clients do not pre-fetch content[2].Thus, in the case of *Cache-and-relay*, R_3 will have to open a stream from the server as soon as it realizes that R_2 has departed and continue downloading from the server for **D** seconds (till it discovers that it can download from R_1). R_3 cannot know *a priori* when R_2 is going to depart. Due to the delays involved in opening a stream from the server, it is quite likely that the playout at R_3 would be disrupted on R_2's departure in the case of *Cache-and-relay*. In the case of *Pre-fetch-and-relay*, if the time required to playout the pre-fetched content is larger than the delay involved in finding a new source to download from, the playout at R_3 would not be disrupted upon R_2's departure from the peer-to-peer network. Pre-fetching content is also advantageous when the download rate is variable. A client can absorb a temporary degradation in download rate without affecting the playout quality if it has sufficient pre-fetched content in its buffer.

A. Control Parameters

In this paper, we analyze the importance and effect of the following three parameters in achieving scalable (in terms of server bandwidth), asynchronous delivery of streams in a peer-to-peer environment through analysis and simulations.

1) $\alpha = \frac{\text{Download rate}}{\text{Playout rate}}$. Without loss of generality, we take the *Playout rate* to be equal to 1 byte/second and, hence the *Download rate* becomes α bytes/second. We assume $\alpha > 1$.

2) T_b : The time it takes to fill the buffer available at a client at the download rate. The actual buffer size at a client is, hence, $\alpha \times T_b$ bytes. The available buffer size at a client limits the time for which a client can download the stream at a rate higher than the playout rate.

3) $\beta = \frac{\text{Future Content}}{\text{Past Content}}$. β represents the ratio of the content yet to be played out, "future content", to the content already played out, "past content", in the buffer.

[2]It can be due to the fact that the playout rate is equal to the download rate or clients may choose not to pre-fetch content.

Next, we discuss the constraints, in terms of α, β and T_b, that must be satisfied for a client to be able to download the stream from the buffer of another client available in the peer-to-peer network.

B. Constraints in the case of an arrival

The following theorem is stated from [23].

Theorem 1: A newly arriving client R_0 can download from the buffer of R_1 if one of the following conditions is satisfied:

- The inter-arrival time between R_1 and R_0 is less than T_b.
- If the inter-arrival time between R_1 and R_0 is greater than T_b, then α should be greater than or equal to 2, R_1 must be over-writing the content in its buffer at the playout rate and the size of the content missing from R_1's buffer should be less than or equal to $\alpha \times T_b$.

The first condition in the above theorem ensures that the content needed by R_0 is present in R_1's buffer. The second condition defines the scenario in which the stream patching technique can be used by R_0. Due to space limitation, please refer to [23] for a detailed discussion.

C. Constraints in the event of a departure

Let us assume that R_0 was streaming from R_1's buffer. R_1 leaves the peer-to-peer network at time $\mathbf{t = t_d}$. If the available buffer size at R_0 is $\alpha \times T_b$ bytes and at $\mathbf{t = t_d}$, the ratio of "future" content to "past" content in R_0's buffer is β, then R_0 has $\left(\frac{\beta\alpha T_b}{1+\beta}\right)$ bytes of the "future" content and $\frac{\alpha T_b}{1+\beta}$ bytes of the "past" content in its buffer. At a *Playout rate* of 1 byte/time unit, R_0 has $\left(\frac{\beta\alpha T_b}{1+\beta}\right)$ time units to find a new source to download from after R_1 departs.

If $\alpha = 1$, then after R_1's departure, R_0 can download from another client R_2's buffer if and only if the content in their buffers overlaps (partially). Fig. 4(a) shows the situation when the buffers of R_0 and R_2 are contiguous. Any client that is ahead of R_0, in terms of playing out the stream, would have some content that R_0 needs to download missing from its buffer and hence, unsuitable for R_0 to download from. Fig. 4(b) depicts such a situation.

With reference to Fig. 4(b), let us assume that the ratio of "future" content to "past" content in R_0's buffer is β and hence, it currently has $\left(\frac{\beta\alpha T_b}{1+\beta}\right)$ bytes of pre-fetched data. Assume that the "missing" content is T_H bytes and that the playout rate is 1 byte/second. If $\alpha > 1$, then R_0 can open two simultaneous streams, one from the server and the other from R_2 , and terminate its stream from the server after it has downloaded the "missing" content and continue to download from R_2 thereafter. Note that for this stream patching technique to work, R_2 should be over-writing contents in its buffer at a rate

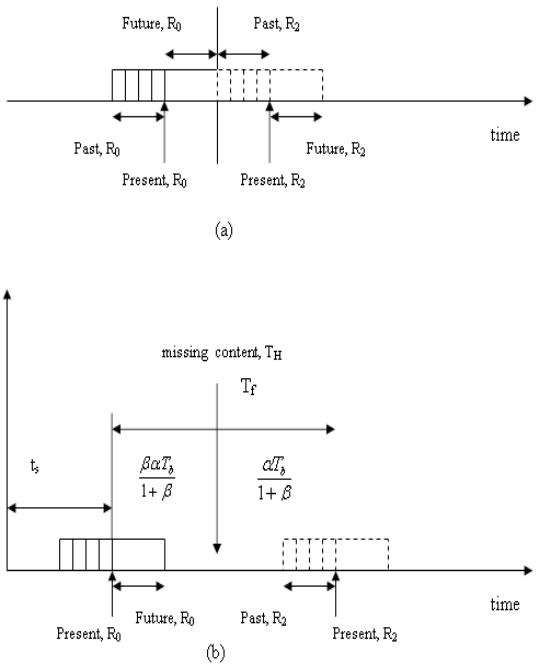

Fig. 4. Possible buffer overlap/non-overlap scenarios for R_0 and R_2.

less than α; in our model we assume that clients over-write the content in their buffer either at the download rate (α) or at the playout rate. Hence, in this case, R_2 should be over-writing its buffer at the rate of 1 byte/second. If this is the case, then R_0 can download from R_2 at the playout rate of 1 byte/second and download the "missing" content from the server at the rate of $(\alpha - 1)$ bytes/second.

The following constraints must be satisfied by the size of the "missing" content, T_H bytes, for R_0 to able to stream from R_2's buffer [23]:

(1) **Constraint imposed due to α:**
The time taken by R_0 to playout the pre-fetched content in its buffer and the "missing" content, T_H bytes, is equal to $(\alpha \times T_b)\left(\frac{\beta}{1+\beta}\right) + T_H$ seconds (at the playout rate of 1 byte/second). The total time needed by R_0 to download T_H bytes from the server at the rate of $(\alpha - 1)$ bytes/second is $\frac{T_H}{\alpha - 1}$ seconds. In order to have the "missing" content available at R_0 before its playout time, the following inequality must be satisfied:

$$(\alpha \times T_b)\left(\frac{\beta}{1+\beta}\right) + T_H \quad \geq \quad \frac{T_H}{\alpha - 1} \qquad (1)$$

The above inequality demands that the time taken to playout the pre-fetched and the "missing" content should be greater than the time taken to download the "missing" content. Note that if $\alpha \geq 2$, then the condition (1) is always satisfied. The stream patching can be used in this case of a departure even when $1 < \alpha < 2$ if a client has sufficient pre-fetched content.

(2) **Constraint imposed by the size of the buffer:**
Suppose that R_0 starts downloading simultaneously from the server and R_2 at time t = t_d. Since it is downloading from R_2 at the playout rate, at any time instant the size of data downloaded from R_2 is exactly equal to the size of the data played out by R_2 after t_d. Thus, to store the content downloaded from the server, R_0 will have to over-write the "past" content in its buffer. Hence, T_H cannot be greater than the size of the "past" content in R_0's buffer at t_d. By our assumption, the "past" content in R_0's buffer at t_d is $\left(\frac{\alpha T_b}{1+\beta}\right)$ bytes and hence,

$$T_H \leq \frac{\alpha T_b}{1+\beta} \qquad (2)$$

III. Server Bandwidth requirement: Analysis

We consider the case of a single CBR media distribution. The playback rate is assumed to be 1 byte/time unit. The client requests are generated according to a Poisson process with rate λ. The time spent by a client downloading the stream is exponentially distributed with rate μ.

Let us assume that a client is able to determine whether it should download the stream from the server or from the buffer of some other client without any delays both in the case of a new arrival as well as in the event of a departure; i.e. we do not take delays like propagation delay and the delay involved in searching the peer-to-peer network for a suitable client to download from into consideration. We later incorporate such delays in Section III-D.

A. Arrivals

A new arrival, R_0, would have to download from the server in either of the following two cases:

- The inter-arrival time between R_0 and the arrival immediately preceding R_0, say R_1, is greater than W; where $W = T_b$ if $1 < \alpha < 2$ or $W = (\alpha \times T_b)\left(\frac{2+\beta}{1+\beta}\right)$ if $\alpha \geq 2$. As mentioned in Theorem 1 in Section II-B, if $\alpha \geq 2$, R_0 can use the stream-patching technique to "catch-up" with R_1 *iff* the size of the content "missing" from R_1's buffer is less than or equal to $\alpha \times T_b$. If R_1 maintains the ratio β in its buffer, then the maximum value of W for R_0 is

$$\begin{aligned} W &= \text{max. "missing" content} \\ &\quad + \text{"past" content at } R_1 \\ &= (\alpha \times T_b) + \left(\frac{\alpha T_b}{1+\beta}\right) \\ &= (\alpha \times T_b)\left(\frac{2+\beta}{1+\beta}\right) \end{aligned}$$

- Suppose R_0 arrives at time $t = t_0$. It can be easily verified that as a consequence of Theorem 1, Section II-B, R_0 can download from only those users that arrived during the interval $T_D = [t_0 - W, t_0)$. If all the users that arrived during the interval T_D have already departed from the peer-to-peer network by $t = t_0$, then R_0 would have to download from the server.

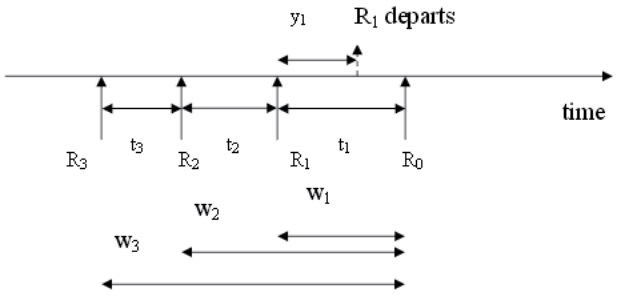

Fig. 5. Timeline illustration of an arrival event (for R_0).

Let w represent the inter-arrival time between any two client requests. Since the arrival process is Poisson with rate λ, the inter-arrival time is exponentially distributed with mean $\frac{1}{\lambda}$, hence:

$$P\{w > W\} = e^{-\lambda W} \tag{3}$$

Suppose a new client request, R_0, arrives at time $t = t_0$. Let N represent the number of arrivals in the interval $T_D = [t_0 - W, t_0)$. Let $N = n$. As mentioned earlier, R_0 would have to download from the server if all the n users that arrived during the interval T_D have departed by $t = t_0$. Let R_i, $i = 1, ..., n$, represent the n users that arrived during the interval T_D. Let y_i be the time spent by client R_i downloading the stream before it departs. Let the inter-arrival time between user R_i and R_0 be w_i. If $y_i \leq w_i$, R_i would have departed by the time R_0 arrives (see Fig. 5). Let **A** represent the event that R_0 has to download from the server because all the users that arrived during the interval T_D departed before R_0's arrival. Then:

$$P\{\text{Event A}|N = n, y_i, w_i\} = P\{N = n\} \prod_{i=1}^{n} P\{y_i \leq w_i\} \times P\{w_i\}$$

where $P\{w_i\}$ is the probability that the inter-arrival time between R_i and R_0 is w_i.

Let the inter-arrival time between R_i and R_{i-1} be t_i with t_1 being the inter-arrival time between R_1 and R_0 and hence, $t_1 = w_1$. Note that because of the memoryless nature of the Poisson arrival process, each t_i is exponentially distributed with mean $\frac{1}{\lambda}$ time units. Thus,

$$\begin{aligned} w_2 &= t_1 + t_2 \\ w_3 &= t_1 + t_2 + t_3 \\ &\vdots \\ w_n &= t_1 + t_2 + \cdots + t_n \end{aligned}$$

Since each w_i $(i \geq 2)$ is a sum of i i.i.d. exponential random variables with mean $\frac{1}{\lambda}$, w_i is an Erlang random variable of order i with mean $\frac{i}{\lambda}$. Hence,

$$P\{w_i\} = \frac{\lambda(\lambda w_i)^{i-1} e^{-\lambda w_i}}{(i-1)!}.$$

Since $w_i \in [0, W)$ $\forall i = 1, \cdots, n$,

$$P\{\text{Event A}\} = \sum_{n=1}^{\infty} \int_0^W \cdots \int_0^W (\textit{Integrand})\ dw_1 \cdots dw_n$$

where the *Integrand* is

$$P\{N = n\} \prod_{i=1}^{n} P\{y_i \leq w_i\} P\{w_i\}$$

To keep the analysis tractable, we solve the above equation for $N = 1$. Then,
$P\{\text{Event A}\}$

$$\begin{aligned} &= \int_0^W P\{N = 1\} \times P\{y_1 \leq w_1\} \times P\{w_1\} dw_1 \\ &= \int_0^W \lambda W e^{-\lambda W} \times (1 - e^{-\mu w_1}) \times \lambda e^{-\lambda w_1} dw_1 \\ &= \lambda W e^{-\lambda W} \times \left(\frac{\mu}{\lambda + \mu} + \frac{\lambda e^{-(\lambda+\mu)W}}{\lambda + \mu} - e^{-\lambda W} \right) \end{aligned} \tag{4}$$

Thus,

$$\begin{aligned} &P\{\text{a new arrival goes to the server}\} \\ &= P\{w > W\} + P\{\text{Event A}\} \end{aligned}$$

where $P\{w > W\}$ and $P\{\text{Event A}\}$ are given by equations (3) and (4), respectively.

B. Departures

Suppose user R_0 is downloading the stream from the buffer of user R_1. Let R_1 depart from the peer-to-peer network at time $t = t_d$ such that by this time R_0 has been downloading and playing out the stream for a duration of t_s time units; i.e. R_0 has been in the peer-to-peer network for t_s time units when R_1 departs. We assume that t_s is long enough for R_0 to have achieved the desired ratio, β, between its "future" and "past" content.

Fig. 4 presents a snapshot of buffer of R_0 and another user, R_2, from whose buffer R_0 can start downloading from instead of going to the server after R_1 has departed. Since we have assumed a sequential access model for client requests, i.e. each client downloads the stream from the beginning and the playout speed is 1 byte/time unit, after R_0 has spent t_s time units downloading the content, its "present" is t_s bytes away from the beginning of the content.

The difference, in terms of number of bytes, between the "present" of R_0 and R_2, represented by T_f in Fig. 4(b), is

$$\begin{aligned} &T_f = \text{"future" content at } R_0 \\ &+\text{"missing" content + "past" content at } R_2 \\ &\Longrightarrow T_f = \left(\frac{\beta \alpha T_b}{1 + \beta} \right) + T_H + \left(\frac{\alpha T_b}{1 + \beta} \right) \end{aligned}$$

The size of the "future" content at R_0 and the "past" content at R_2 have been calculated using the assumption that both R_0 and R_2 maintain the ratio β. As discussed in Section II-C the download rate α and the buffer size impose certain constraints

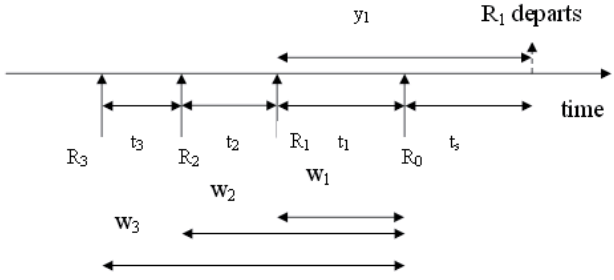

Fig. 6. Timeline illustration of a departure event (for R_1).

on the size of the "missing" content, T_H, for R_2 to be suitable for R_0 to download from. The maximum value of T_f is

$$T_f = \begin{cases} (\alpha \times T_b)\left(\frac{\beta-\alpha+2}{(1+\beta)(2-\alpha)}\right) & \text{if } \alpha \leq \left(\frac{2+\beta}{1+\beta}\right) \\ (\alpha \times T_b)\left(\frac{2+\beta}{1+\beta}\right) & \text{otherwise} \end{cases} \quad (5)$$

We provide a detailed derivation of Equation 5 in the appendix.

After the departure of R_1, only those clients whose "present" is ahead of R_0's "present" by a value less than or equal to the maximum value of T_f are suitable for R_0 to start downloading from, assuming that all such clients maintain the same ratio β in their buffers.

Since we have assumed the playout speed to be 1 byte/second, a client whose "present" is T_f bytes ahead of R_0 must have arrived T_f time units before R_0. Hence, the constraint on the suitability of a client (mentioned in the preceding paragraph) can be re-stated as: *on R_1's departure R_0 can download from only those clients that arrived at most T_f time units before R_0; where the value of T_f is given by (5).*

Suppose that the client request R_0 arrives at $t = t_0$. Let N represent the number of arrivals in the interval $[t_0 - T_f, t_0)$. Let $N = n$ and $R_i, i = 1, \cdots, n$ be the n client requests. Suppose R_0 starts downloading the stream from R_1 and R_1 departs the peer-to-peer network at $t_d = t_0 + t_s$. Thus, R_0 has spent t_s time units downloading the stream when R_1 departs. Let w_i be the inter-arrival time between R_i and R_0 and y_i be the time spent by R_i downloading the stream (see Fig. 6). If all the clients $R_i, i = 2, \cdots, n$ have departed by the time $t_d = t_0 + t_s$, i.e. $y_i \leq t_s + w_i$, then R_0 would have no option but to download from the server on R_1's departure. Let event **B** represent the situation that R_0 downloads from the server on R_1's departure. Then,

$$P\{\text{Event B}|N = n, t_s, y_i, w_i\} = P\{N = n\}P\{t_s\}\prod_{i=2}^{n} P\{w_i\}P\{y_i \leq t_s + w_i\}$$

where $P\{t_s\}$ is the probability that R_0 has downloaded the stream for t_s time units by the time R_1 departs, $P\{w_i\}$ is the probability that the inter-arrival time between R_i and R_0 is w_i and $i = 2, \cdots, n$.

To keep the analysis tractable, let us assume $N = 2$; this represents the scenario where R_0 starts downloading from R_1 on arrival and could potentially download from R_2 on R_1's departure. We have,

$$P\{\text{Event B}\} = \int_0^{\infty}\int_0^{T_f} (\textit{Expression})\ dw_2 dt_s \quad (6)$$

where *Expression*[3] is:

$$\left(\frac{e^{-\lambda T_f}(\lambda T_f)^2}{2!}\right)(\mu e^{-\mu t_s})\left(\frac{\lambda(\lambda w_2)e^{-\lambda w_2}}{1!}\right)(1 - e^{-\mu(t_s+w_2)})$$

C. Server Load

Let event **S** represent the situation that a client request downloads the stream from the server. From the preceding discussion in Section III-A and III-B,

$$P\{\text{Event S}\} = P\{w > W\} + P\{\text{Event A}\} + P\{\text{Event B}\}$$

where the right-hand terms are given by equations (3), (4) and (6), respectively.

In our model we have assumed that client requests are generated according to a Poisson process with rate λ. Hence, in steady state, i.e. after there are enough end-hosts in the peer-to-peer network, the average number of client requests that download the stream from the server is $\lambda \times P\{\text{Event S}\}$.

D. Incorporating the delays

Let us revisit the scenario discussed in Section II-C. Suppose that R_2 is a suitable client for R_0 to start downloading from after R_1's departure but it takes R_0 D time units after R_1's departure to determine this. R_0 needs to know what is available on each client's buffer and process that information to determine the suitability of each client. There will also be some propagation delay involved in the transmission of the *meta-data*[4] traffic amongst clients in the overlay network.

At the time of R_1's departure, R_0 has $\left(\frac{\beta\alpha T_b}{1+\beta}\right)$ time units of "future" data. We assume that if R_0 is not able to find R_2 by the time it finishes playing out its "future" content ($D > \frac{\beta\alpha T_b}{1+\beta}$), R_0 starts downloading from the server. If $D \leq \left(\frac{\beta\alpha T_b}{1+\beta}\right)$, R_0 can absorb the delay without any disruption of its playout. In the next section, we present analytical results after incorporating the delay involved in the event of a departure into our model assuming that the delay is uniformly distributed.

IV. Performance Evaluation

In this section, we evaluate the performance of *Pre-fetch-and-relay* based on the analytical model presented in this paper and compare it with simulation results. We also present analytical and simulation results after incorporating the various delays into our model (as discussed in Section III-D). We also

[3]The integrand for computing $P\{\text{Event B}\}$ is computed assuming that R_0 starts downloading from R_1 on arrival.

[4]The messages exchanged in determining the clients who are still present in the peer-to-peer network and their buffer contents.

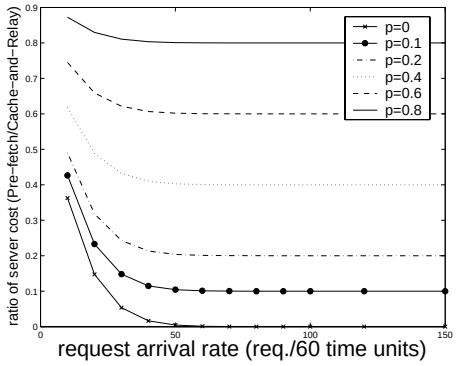

Fig. 7. Importance of Pre-fetched content

compare the performance of *Pre-fetch-and-relay* with *Cache-and-relay* with respect to savings in server bandwidth. We refer to the protocols oStream [6] and OSMOSIS [17] by the generic term *Cache-and-relay* because they correspond to the situation when $\alpha = 1$ (hence, a client cannot pre-fetch any content).

A. Advantage of Pre-fetching

If a client does not *pre-fetch* content (as in *Cache-and-relay*), then on premature departure of its current source it has no option but to start downloading from the server till the time it is able to locate another source in the peer-to-peer network. In this section, we analytically compare the performance of our *Pre-fetch-and-relay* scheme against the *Cache-and-relay* scheme proposed in [6].

Suppose R_0 starts downloading the stream from R_1 upon arrival. Let the inter-arrival time between R_0 and R_1 be w_1 and the time spent by R_i $(i = 0, 1)$ downloading the stream be t_i. Under the *Cache-and-relay* scheme, R_0 will have to start downloading from the server at some point if R_1 departs before R_0. In both *Cache-and-relay* as well as *Pre-fetch-and-relay*, a client that is forced to download from the server after the premature departure of its source can stop downloading from the server after it has found another source in the peer-to-peer network. To keep the analysis tractable, we have not considered this "optimization" in this analysis. But since this "optimization" technique to reduce the server bandwidth is possible under both schemes, the trend exhibited by the results presented in this section would remain the same. Using the same analytical model of Section III,

$$\begin{aligned} &P\{R_1 \text{ departs before } R_0\} \\ &= P\{t_1 \leq t_0 + w_1\} \times P\{t_0\} \times P\{w_1\} \\ &= (1 - e^{-\mu(t_0+w_1)}) \times \mu e^{-\mu t_0} \times \lambda e^{-\lambda w_1} \end{aligned}$$

Thus, the probability that R_0 will have to download from the server under the *Cache-and-relay* scheme due to the premature departure of its current source, represented by P_d, is

$$P_d = \int_0^W \int_0^\infty (1 - e^{-\mu(t_0+w_1)}) \times \mu e^{-\mu t_0} \times \lambda e^{-\lambda w_1} dt_0 dw_1$$

In the *Pre-fetch-and-relay* scheme, as calculated in Section III, the probability that R_0 downloads from the server after the premature departure of its current source is given by $P\{\text{Event B}\}$. Note that in computing $P\{\text{Event B}\}$ we had not taken the various delays mentioned in Section III-D, into consideration. Let p represent the probability that R_0 is not able to locate a new source to download from, after R_1's departure, before it finishes playing out its *pre-fetched* content. R_0 would be forced to start downloading from the server in such a scenario. Thus, under the *Pre-fetch-and-relay* scheme, the probability that R_0 will download from the server due to the premature departure of its current source, represented by P_F, is

$$P_F = P_d \times P\{\text{Event B}\} + P_d \times (1 - P\{\text{Event B}\}) \times p$$

where the first term represents the scenario where there are no new sources in the peer-to-peer network for R_0 to download from after the departure of its current source, and the second term represents the situation where R_0 finishes playing out its *pre-fetched* content before it is able to locate a new source.

The probability p can be thought of as the fraction of time when a client does not have enough *pre-fetched* content to be able to locate a new source after the departure of its current source. Fig. 7 compares the performance of *Cache-and-relay* and *Pre-fetch-and-relay* for different values of p. Note that P_F degenerates to P_d for $p = 1$. The ratio of the server bandwidth requirement for *Pre-fetch-and-relay* to the server bandwidth requirement for *Cache-and-relay* is plotted along the y-axis. The value of α is 2 and β is 100,000. The buffer size is 10 time units. It is evident from the figure that in the presence of client departures, *pre-fetching* "sufficient" content by clients can help reduce the server bandwidth requirement significantly.

B. Simulation Model

Table I presents the settings of the various parameters used to obtain the results presented in this section.

Figure	Buffer size	β	$(1/\mu)$	Delay
8	5	100,000	1000	No
9	10	100,000	1000	No
10	20	100,000	1000	No
11	10	100,000	100	No
12	10	1	1000	Yes
13	10	100,000	1000	Yes

TABLE I
SETTINGS OF VARIOUS PARAMETERS USED IN SIMULATIONS.

The quantity $(1/\mu)$ represents the average time spent by a client downloading the stream. A "**No**" in the column labeled "Delay" indicates that the various delays discussed in Section III-D were not considered in calculating the server bandwidth requirement; a "**Yes**" indicates that the delays were considered.

The plots on the left-hand-side of Figures 8, 9, and 10 show the server bandwidth requirements obtained from our analytical model for the simplified cases when $N = 1$ in Section III-A and $N = 2$ in Section III-B. The plots on the right-hand-side show the corresponding simulation results.

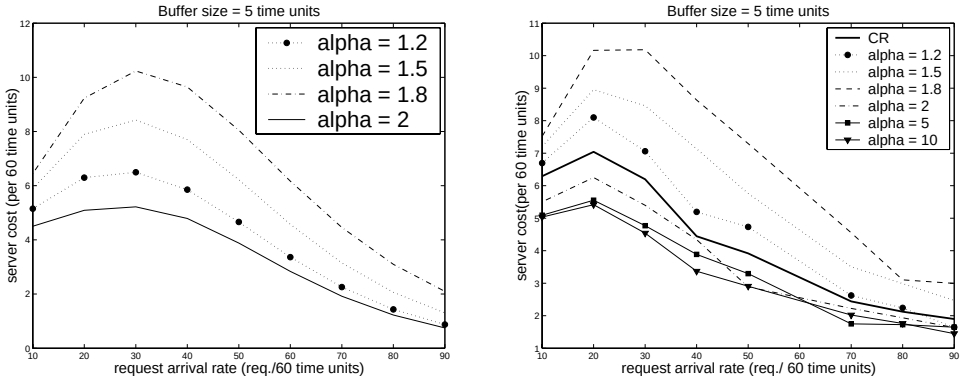

Fig. 8. Analysis (left) and simulation (right) when mean download time = 1000 time units and buffer size = 5 time units.

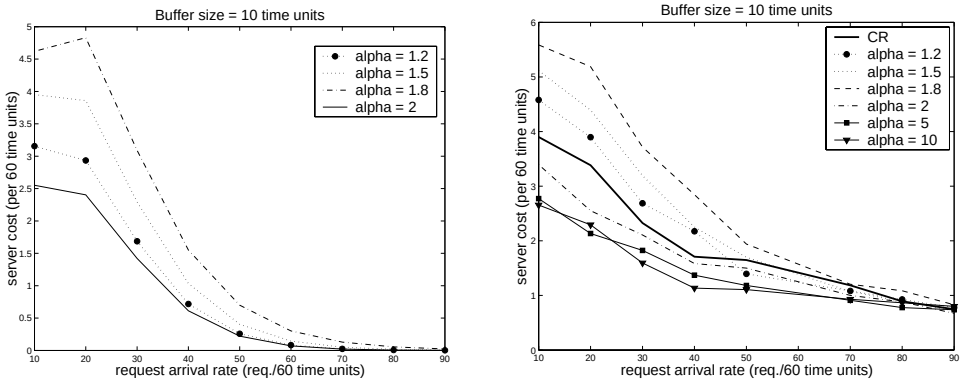

Fig. 9. Analysis (left) and simulation (right) when mean download time = 1000 time units and buffer size = 10 time units.

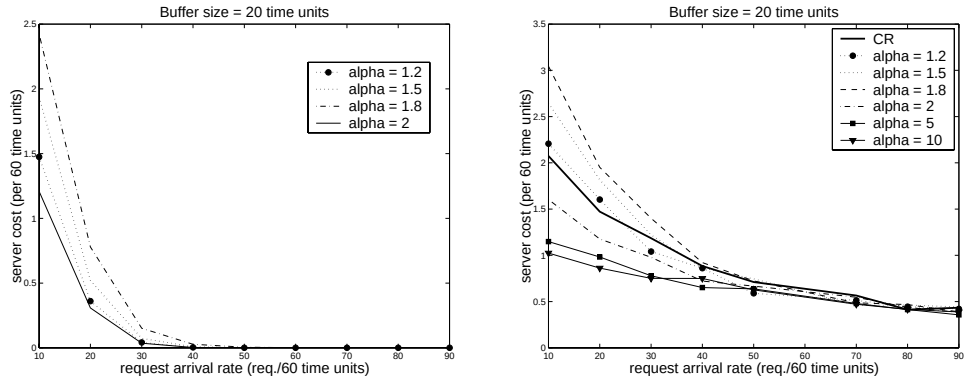

Fig. 10. Analysis (left) and simulation (right) when mean download time = 1000 time units and buffer size = 20 time units.

All of these results are under an assumption of Delay=No. This assumption is relaxed in the results shown in Figures 12 and 13. Namely, we assume that the total delay involved in switching the streaming session to another client after a departure is uniformly distributed over the interval [0,9]. With a buffer size of 10 time units and $\beta = 100,000$, a client will have 9.999 time units of "future" content in its buffer after it achieves the ratio β whereas with $\beta = 1$ it will have only 5 time units of "future" content.

In all simulations, the total number of client arrivals was set at 3,000. Each point on a graph represents an average over 10 independent runs.

C. Summary of Observations

If the download rate is sufficiently high, $\alpha \geq 2$, dPAM has an advantage over *Cache-and-relay* in reducing the server bandwidth when the resources available for overlay stream multicast are constrained, for example when the buffer size is

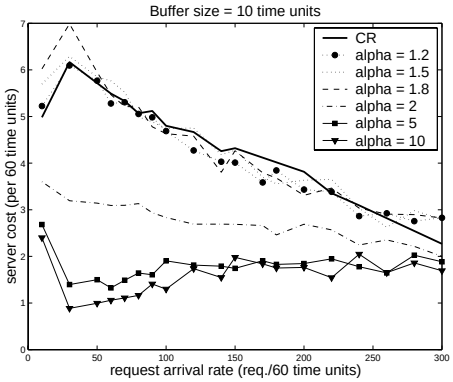

Fig. 11. Simulation results when mean download time = 100 time units and buffer size = 10 time units.

small or when the request arrival rate is low. The advantage stems from the fact that a higher download rate enables a client to open two simultaneous connections for a short duration to "catch-up" with the buffer of another client using the technique of stream patching. This advantage is more pronounced for higher client departure rate. If clients depart frequently from the peer-to-peer network, it reduces the caching capacity of the peer-to-peer network, thus patching content from the server becomes more beneficial. As the buffer size and the request arrival rate increase, the advantage of our dPAM protocol over *Cache-and-relay* is mitigated and for a given buffer size, at a sufficiently high request arrival rate, *Cache-and-relay* matches the performance of dPAM in terms of server bandwidth even when the download rate is very high.

When $1 < \alpha < 2$, *Pre-fetch-and-relay* leads to a greater server load than *Cache-and-relay* for small arrival rates. As we increase α, the time taken to fill the buffer at download speed, T_b decreases. For example, for a buffer size of 5 time units, for *Cache-and-relay* ($\alpha = 1$), T_b = 5; whereas when α=1.8, T_b=2.78. Thus, in the former case, a new arrival can reuse the stream from someone who arrived at most 5 time units earlier whereas in the latter case a new arrival can download from someone who arrived at most 2.78 time units earlier. Hence, in the latter case, more new arrivals have to download from the server. This effect can be mitigated by increasing the buffer size and also for increasing client arrival rate.

The results presented in this section also show that as the available buffer at the client increases, the required server bandwidth to support a particular request arrival rate decreases, both in *Cache-and-relay* as well as *Pre-fetch-and-relay* (for all values of α). This observation is in agreement with the results obtained in [6].

The amount of time that clients spend downloading a stream is an important factor in determining server bandwidth requirements. Peer-to-peer network based asynchronous media content distribution is suited for situations in which the content being distributed is large; so that the end-hosts participating in the peer-to-peer network are available for a long time. In a scenario where end-hosts keep departing after a short interval, the server load can be considerably high due to the fact that a lot of requests may have to start downloading from the server due to the departure of clients they were downloading from. Fig. 11 presents the simulation results when the mean time spent by a client downloading the stream ($1/\mu$) is 100 time units. Compared to the case when $1/\mu = 1000$, the server bandwidth requirement is considerably higher even for very high client arrival rates.

We refer the reader to [23] for a more detailed discussion on the simulation results presented in Figures 8(right), 9(right), 10(right) and 11.

The server bandwidth requirements obtained from our analytical model (Figures 8(left), 9(left), 10(left)) display the same trends as observed through simulations. The analytical results are more optimistic than the results obtained through simulations because of the assumption that clients are able to achieve the ratio β in the buffer before they are forced to switch their streaming session because of client departures. If the ratio between the "future" and the "past" content in a client's buffer is less than β, then not only does it have less time to discover another client to download from but the number of suitable clients available in the peer-to-peer network is also reduced because of a smaller T_f.[5]

Figures 12 and 13 present the results from analysis and simulations after incorporating the various delays into our model. In Figure 12, with $\beta = 1$, clients have 5 time units of "future" content whereas the delays involved are uniformly distributed over the interval [0,9]. Hence, in a significant number of cases, a client would be forced to download the stream from the server after a departure because it would be unable to find another client to download from before it finishes playing out the "future" content. As a result, the server bandwidth requirement keeps on increasing even for high client arrival rate. In Figure 13, with $\beta = 100,000$, clients have 9.99 time units of "future" content. Hence, clients are able to absorb the delays without any disruption to their playout and are not forced to download from the server. As a result, the server bandwidth requirement displays the same trend as observed when delays were not considered. These results aim to underscore the importance of taking advantage of a higher download rate to *pre-fetch* content in achieving a better playout performance and lower server bandwidth requirement.

V. Implementation Issues

There are two main components to dPAM: (1) Buffer Management: How should a node manage its buffer? and (2) Content Location: How does a node locate a set of potential sources from which to prefetch its content upon arrival, or upon being disconnected due to the premature departure of its current source? In this paper, we have focused on the buffer management component of dPAM. In this section, we briefly outline how the content location component of dPAM could be readily implemented.

First, we note that the content cached in any P2P node is uniquely identifiable by the absolute time of the first byte of the feed currently being played out at that node. For example, if at time t a node i is playing out the d^{th} second of a feed

[5]As discussed in Section III-B, the size of T_f is related to the "future" and the "past" content in the buffers of the clients.

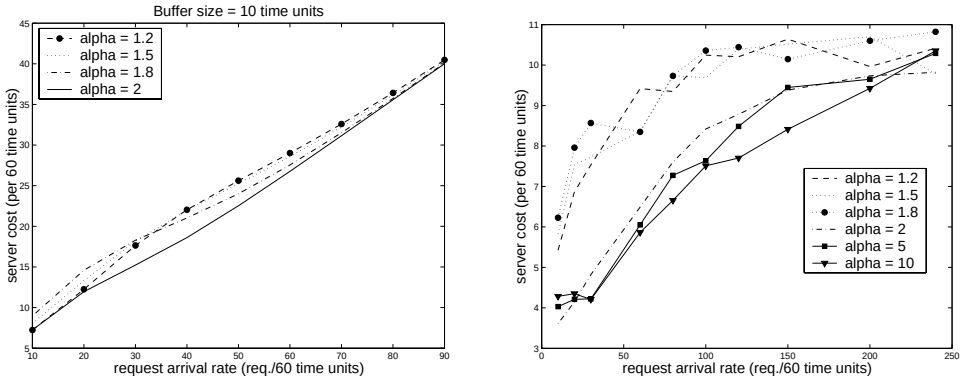

Fig. 12. Analysis (left) and simulation (right) *with delay* when mean download time = 1000 time units and buffer size = 10 time units, $\beta = 1$

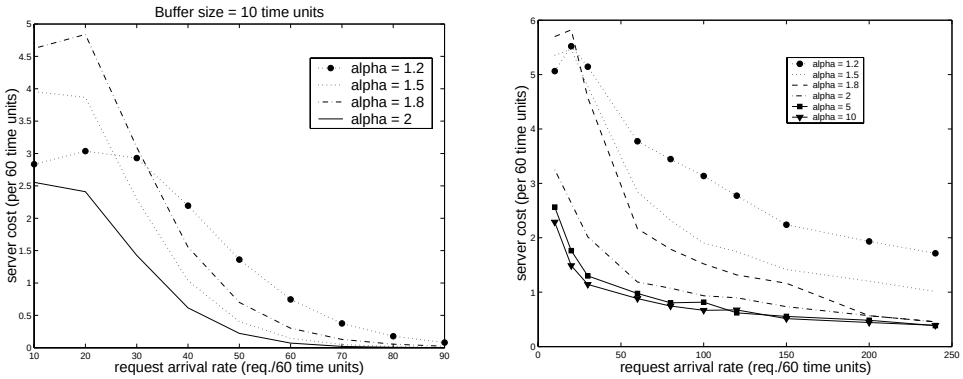

Fig. 13. Analysis (left) and simulation (right) *with delay* when mean download time = 1000 time units, buffer size = 10 time units, $\beta = 100,000$

f, then the content of node i storage at any point in time is uniquely identified by the string $f@t_0$ where $t_0 = t - d$. Thus one can treat $f@t_0$ as the unique reference string for the content available at node i. Similarly, it also follows that a node j searching for content could uniquely generate the reference string for the cached content it is seeking by specifying the name of the feed and its own starting time for that feed.

Our ability to uniquely name the cache content at any node in the system gives us a simple mechanism for implementing dPAM's content location component. Specifically, this could be done using any number of existing P2P content location protocols, ranging from widely deployed controlled flooding protocols (e.g., gnutella [1]) to more scalable DHT-based protocols (e.g., CAN [21] and CHORD [25]), or recently proposed hybrids thereof [10].

Upon joining (leaving) the asynchronous multicast of a feed f, a node would advertise the availability of its cached portion of f by adding (removing) the appropriate reference string into (from) the pool of available content (this is akin to a node adding/removing a file to/from the set of files it is contributing to a P2P file sharing application). When a node needs to locate a new source of content, it would query the system using an appropriately formed reference string (possibly including wild-cards to allow for ranges, etc.) In our dPAM scheme, a node arriving at t_0 needs to query a string range that spans the arrival times $[t_0 - W, t_0)$. If matches exist (or are locatable using the content location protocol), that node would receive a list of all candidate sources, possibly with additional meta information coded into the matching reference strings (e.g., load information, network distance, etc.) From such a candidate list, the node would select the most appropriate source. If matches do not exist, or if the candidate list is not responsive (perhaps due to delays in propagating departures through the P2P name space), the node will resort back to the server.

VI. Related Work

Delivery of streams to asynchronous clients has been the focus of many studies, including periodic broadcasting [26], [14], [12], [18] and stream patching/merging techniques [5], [11], [7], [8]. In periodic broadcasting, segments of the media object (with increasing sizes) are periodically broadcasted on dedicated channels, and asynchronous clients join one or more broadcasting channels to download the content. The approach of patching [13] allows asynchronous client requests to "catch up" with an ongoing multicast session by downloading the missing portion through server unicast. In merging techniques [9], clients merge into larger and larger multicast session repeatedly, thus reducing both the server bandwidth and the

network link cost. These techniques rely on the availability of a multicast delivery infrastructure at the lower level.

The idea of utilizing client-side caches has been proposed in several previous work [24], [20], [15]. The authors of [6], propose an overlay, multicast strategy, *oStream*, that leverages client-side caching to reduce the server bandwidth as well the network link cost. Assuming the client arrivals to be Poisson distributed, they also derive analytical bounds on the server bandwidth and network link cost. However, this work does not consider the effect of the departure of the end-systems from the overlay network on the efficiency of overlay multicast. As mention earlier, *oStream*, does not consider the effect of streaming rate, it is a *Cache-and-relay* strategy, and hence, does not incorporate patching techniques to reduce server bandwidth when the download rate is high. The main objective of the protocol, *OSMOSIS*, proposed in [17] is to reduce the network link cost. The effect of patching on server load has not been studied.

A different approach to content delivery is the use of periodic broadcasting of encoded content as was done over broadcast disks [2] using IDA [19], and more recently using the Digital Fountain approach which relies on Tornado encoding [4], [3]. These techniques enable end-hosts to reconstruct the original content of size *n* using a subset of any *n* symbols from a large set of encoded symbols. Reliability and a substantial degree of application layer flexibility can be achieved using such techniques. But these techniques are not able to efficiently deal with real-time (live or near-live) streaming media content due to the necessity of encoding/decoding rather large stored data segments.

VII. Conclusion and Future Work

We proposed dPAM, a *pre-fetch-and-relay* protocol that allows a peer to serve as a source for other peers, while prefetching a portion of the stream ahead of its playout time. In contrast to existing cache-and-relay schemes, dPAM is more scalable in highly dynamic P2P systems. This is because a departure of a peer does not necessarily force its children peers (for whom it is serving as source) to go to the original server. Rather a child peer can continue its playout uninterrupted from its prefetched data until it discovers a new source-peer. Through mathematical analysis and extensive simulations, we show that, if the download rate is sufficiently greater than the playout rate ($\alpha \geq 2$), our distributed prefetching scheme significantly reduces the load on the server as it effectively increases the capacity of the P2P system. At the same time, clients can achieve a better playout performance.

One aspect of dPAM that we did not evaluate (but should be evidently obvious) is that the prefetching buffer allows a client to withstand, not only the departure of source-peers, but also network jitters associated with the bandwidth from the source. Indeed, buffering (via prefetching) is commonly used in streaming media players for that purpose, suggesting that the same storage capacity at a client could be used for smoothing network jitters *as well as* improving the scalability of P2P asynchronous multicast.

In this paper, we have assumed that a client starts the process of replacing a source-peer upon discovering the departure of such a source, and that the replenishment of the client's buffer is done by contacting only one source-peer. Clearly, a client can minimize the delay experienced in finding a new peer to download from after the departure of its source-peer by pre-computing and maintaining a list of other potential source-peers. In the event of the departure of its source-peer, a client can reduce its search time for a new peer by first checking out the peers on its list. Also, when replinishing its prefetched buffer, a client may be able to leverage multiple sources concurrently [22].

More importantly, a client can proactively switch from one source-peer to another in order to reduce the transmission delay of its download or to optimize the overall network link cost. To that end, maintaining a list of potential source-peers can also help reduce the delay in finding the "optimal" source- peer. In effect, one may think of this process as a distributed optimization process whereby each peer in the P2P asynchronous multicast overlay is performing local optimization (by proactively selecting its source from many potential candidates). Currently, we are exploring the impact of the above mentioned optimizations on the performance of our protocol. This includes issues of convergence and stability.

Appendix

As discussed in Section II-C the download rate α and the buffer size impose certain constraints on the size of the "missing" content, T_H, for R_2 to be suitable for R_0 to download from (see Fig. 4). If $\alpha \geq 2$, condition (1), Section II-C is always satisfied and hence, the maximum size of the "missing" content is given by condition (2), Section II-C. Hence, when $\alpha \geq 2$, the maximum value of T_f, assuming that both R_2 and R_0 maintain the ratio β in their buffers, is

$$\begin{aligned} T_f &= \left(\frac{\beta\alpha T_b}{1+\beta}\right) + \left(\frac{\alpha T_b}{1+\beta}\right) + \left(\frac{\alpha T_b}{1+\beta}\right) \\ \Longrightarrow T_f &= (\alpha \times T_b)\left(\frac{2+\beta}{1+\beta}\right) \end{aligned}$$

Now consider the case of $1 < \alpha < 2$. Condition (1), Section II-C can be restated as

$$T_H \leq \left(\frac{\alpha\beta T_b}{1+\beta}\right)\left(\frac{\alpha-1}{2-\alpha}\right)$$

We can derive the condition on α that determines whether condition (1), Section II-C or condition (2), Section II-C restricts the maximum size of the "missing" content; condition (1), Section II-C determines the maximum size of the missing content *iff*

$$\begin{aligned} \left(\frac{\alpha\beta T_b}{1+\beta}\right)\left(\frac{\alpha-1}{2-\alpha}\right) &\leq \left(\frac{\alpha T_b}{1+\beta}\right) \\ \Longrightarrow \alpha &\leq \left(\frac{2+\beta}{1+\beta}\right) \end{aligned} \tag{7}$$

If (7) is satisfied then the maximum value of T_f, assuming that both R_2 and R_0 maintain the ratio β in their buffers, is

$$\begin{aligned} T_f &= \left(\frac{\beta\alpha T_b}{1+\beta}\right) + \left(\frac{\alpha\beta T_b}{1+\beta}\right)\left(\frac{\alpha-1}{2-\alpha}\right) + \left(\frac{\alpha T_b}{1+\beta}\right) \\ &= (\alpha \times T_b)\left(\frac{\beta-\alpha+2}{(1+\beta)(2-\alpha)}\right) \end{aligned}$$

REFERENCES

[1] Gnutella. http://www.gnutella.com.

[2] A. Bestavros. AIDA-based Real-time Fault-Tolerant Broadcast Disks. In *Proceedings of IEEE RTAS'96*, May 1996.

[3] J. Byers, J. Considine, M. Mitzenmacher, and S. Rost. Informed Content Delivery across Adaptive Overlay Networks. In *Proceedings of ACM SIGCOMM*, 2002.

[4] J. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A Digital Fountain Approach to Reliable Distribution of Bulk Data. In *Proceedings of ACM SIGCOMM*, 1998.

[5] S. W. Carter and D. D. E. Long. Improving Video On-demand Server Efficiency through Stream Tapping. In *Proceedings of IEEE International Conference on Computer Communication and Networks (ICCN)*, 1997.

[6] Y. Cui, B. Li, and K. Nahrstedt. oStream: Asynchronous Streaming Multicast in Application-Layer Overlay Networks. *IEEE Journal on Selected Areas in Communications*, 22(1), January 2004.

[7] D. Eager, M. Vernon, and J. Zahorjan. Minimizing Bandwidth Requirements for On-demand Data Delivery. In *Proceedings of Workshop on Multimedia and Information Systems(MIS)*, 1998.

[8] D. Eager, M. Vernon, and J. Zahorjan. Bandwidth Skimming: A technique for Cost-Efficient Video On-demand. In *Proceedings of ST/SPIE Conference on Multimedia Computing and Networking (MMCN)*, 2000.

[9] D. Eager, M. Vernon, and J. Zahorjan. Minimizing Bandwidth Requirements for On-Demand Data Delivery. *IEEE Transactions on Knowledge and Data Engineering*, 13(5), 2001.

[10] P. Ganesan, Q. Sun, and H. Garcia-Molina. YAPPERS: A Peer-to-peer Lookup Service Over Arbitary Topology. In *Proceedings of IEEE Infocom*, 2003.

[11] L. Gao and D. Towsley. Supplying Instantaneous Video On-demand Services using Controlled Multicast. In *Proceedings of IEEE International Conference on Multimedia Computing and Systems (ICMCS)*, 1999.

[12] A. Hu. Video-on-demand Broadcasting Protocols: A Comprehensive Study. In *Proceedings of IEEE INFOCOM*, April 2001.

[13] K. Hua and Y. Cai amd S.Sheu. Patching:A Multicast Technique for True On-Demand Services. In *Proceedings of ACM Multimedia*, 1998.

[14] K. A. Hua and S. Sheu. Skyscraper broadcasting: A new broadcasting scheme for metropolitan video-on-demand systems. In *Proceedings of ACM SIGCOMM*, September 1997.

[15] K. A. Hua, D. A. Tran, and R. Villafane. Caching Multicast Protocol for On-demand Video Delivery. In *Proceedings of ST/SPIE Conference on Multimedia Computing and Networking (MMCN)*, 2000.

[16] S. Jin and A. Bestavros. Scalability of Multicast Delivery for Non-sequential Streaming Access. In *Proceedings of SIGMETRICS'2002: The ACM International Conference on Measurement and Modelling of Computer Systems*, 2002.

[17] S. Jin and A. Bestavros. OSMOSIS: Scalable Delivery of Real-Time Streaming Media in Ad-Hoc Overlay Networks. In *Proceedings of IEEE ICDCS'03 Workshop on Data Distribution in Real-Time Systems*, 2003.

[18] A. Mahanti, D. Eager, M. Vernon, and D. Sundaram-Stukel. Scalable On-demand Media Streaming with Packet Loss Recovery. In *Proceedings of ACM SIGCOMM*, 2001.

[19] M. O. Rabin. Efficient Dispersal of Information for Security, Load Balancing and Fault Tolerance. In *Journal of the Association for Computing Machinery*, volume 36, pages 335–348, April 1997.

[20] S. Ramesh, I. Rhee, and K. Guo. Multicast with Cache (mcache): An Adaptive Zero-delay Video On-demand. In *Proceedings of IEEE INFOCOM*, 2001.

[21] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. CAN: A Scalable Content-Addressable Network. In *Proceedings of ACM SIGCOMM*, September 2001.

[22] P. Rodriguez and E. W. Biersack. Dynamic Parallel-Access to Replicated Content in the Internet. *IEEE/ACM Transactions on Networking*, August 2002.

[23] A. Sharma, A. Bestavros, and I. Matta. dPAM: A Distributed Prefetching Protocol for Scalable, Asynchronous Multicast in P2P Systems. In *BU-CS Technical Report*, June 2004.

[24] S. Sheu, K. Hua, and W. Tavanapong. Chaining: A Generalized Batching technique for Video On-demand. In *Proceedings of IEEE International Conference on Multimedia Computing and Systems (ICMCS)*, 1997.

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

[26] S. Viswanathan and T. Imielinski. Pyramid Broadcasting for Video On-demand Service. In *Proceedings of ST/SPIE Conference on Multimedia Computing and Networking (MMCN)*, 1995.