Performance Evaluation of Distributed Prefetching for Asynchronous Multicast in P2P Networks*

Abhishek Sharma[1], Azer Bestavros[2], and Ibrahim Matta[2]

[1] Elec. & Comp. Eng., Boston University, Boston, MA 02215, USA abhishek@bu.edu
[2] Comp. Sc., Boston University, Boston, MA 02215, USA {best,matta}@cs.bu.edu

Abstract. We consider the problem of delivering real-time, near real-time and stored streaming media to a large number of asynchronous clients. This problem has been studied in the context of asynchronous multicast and peer-to-peer content distribution. In this paper we evaluate through extensive simulations the performance of the distributed *prefetching* protocol, dPAM [20], proposed for scalable, asynchronous multicast in P2P systems. We show that the *prefetch-and-relay* strategy of dPAM can reduce the server bandwidth requirement quite significantly, compared to the previously proposed *cache-and-relay* strategy, even when the group of clients downloading a stream changes quite frequently due to client departures.

1 Introduction

On-demand media distribution is fast becoming an ubiquitous service deployed over the Internet. The long duration and high bandwidth requirements of streaming media delivery present a formidable strain on server and network capacity. Hence, scalable delivery techniques, both in terms of network link cost as well as server bandwidth requirement, are critical for the distribution for highly popular media objects.

For the delivery of real-time media to synchronous requests, multicast solutions (whether using network support in case of IP multicast or using end-system support through peer-to-peer networks) are attractive as they reduce both network link costs and server bandwidth requirements for serving a large number of clients [22,5,13,11]. However, a number of scenarios can be envisioned in which the client requests for streaming media objects are likely to be *asynchronous*. This is true for requests to stored streaming media objects (e.g., on-demand delivery of popular movie clips or news briefs to clients), as well as for requests to buffered live streams (e.g., playout of a webcast to a large number of clients requesting that webcast asynchronously but within a short interval).

To enable asynchronous access to streaming media objects, various IP multicast based periodic broadcasting and stream merging techniques [22,13,11,17]

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

C.-H. Chi, M. van Steen, and C. Wills (Eds.): WCW 2004, LNCS 3293, pp. 154–168, 2004.

have been proposed. These techniques are scalable in terms of network link cost by virtue of multicast messaging. To achieve scalability in terms of server bandwidth requirement, they try to ensure that a relatively small number of multicast sessions (possibly coupled with short unicast sessions) are enough to cater to a large number of asynchronous client requests. The assumption about the availability of a network infrastructure supporting IP multicast may be practical within the boundary of a multicast-enabled intranet, but it is yet to become an ubiquitous alternative in today's Internet. This realization has led to an alternate approach of using application-layer (or end-system) multucast.

Application-layer multicast, or overlay multicast, can facilitate the deployment of multicast-based applications in the absence of IP multicast [5]. Multicast can be achieved in overlay networks through data relay among overlay members via unicast. Apart from elevating the multicast functionality to the application layer, this approach also provides a substantial degree of flexibility due to the fact that each node in an overlay network can perform more complicated application-specific tasks which might be too expensive to perform at the routers in case of IP multicast.

1.1 Paper Contribution

In this paper, we evaluate, through extensive simulations, the impact of streaming rate and the departure of nodes on the scalability of the *prefetch-and-relay* strategy employed in the dPAM protocol [20], proposed for scalable asynchronous overlay multicast. We highlight the importance of "prefetching" content in achieving a better playout quality in a scenario where client nodes participating in the overlay network depart from the network (or stop downloading the stream). We refer the reader to [20] for a detailed description of dPAM and its analysis.

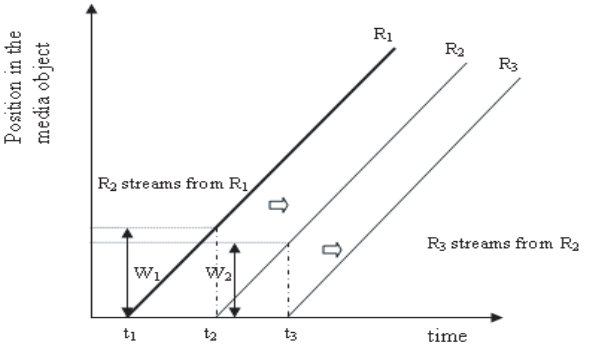

Fig. 1. Overlay-based asynchronous streaming

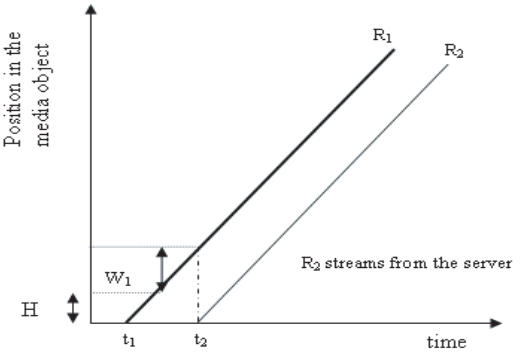

Fig. 2. Overlay-based aynchronous streaming

2 Prefetch-and-Relay

In this section, we review the *prefetch-and-relay* strategy employed in the distributed prefetching protocol, dPAM [20], proposed for scalable, asynchronous multicast in P2P networks. We illustrate the asynchronous delivery of streams through overlay networks using 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 Figure 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 Figure 1, leveraging the caches at end-hosts helps to serve three clients using just one stream from the server.

In Figure 2, by the time 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 Figure 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) 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 [12]. 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 $\alpha - 1 \geq 1 \Rightarrow \alpha \geq 2$. Thus, we need the total download rate to be at least twice the playback rate for stream patching to work for a new arrival.

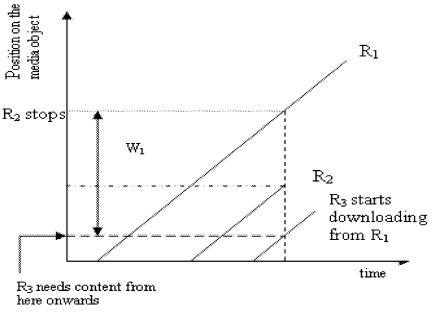

Fig. 3. Switching streams on a departure: streaming from another client

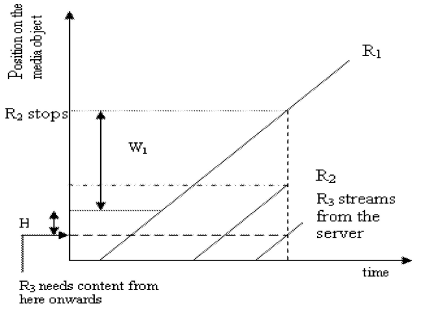

Fig. 4. Switching streams on a departure: streaming from the server

A request may have to switch its streaming session under certain situations. As shown in Figure 3, R_3 initially streams from R_2 until R_2 leaves the overlay network. Since the content R_3 needs is still availabe in R_1's buffer, R_3 starts streaming from R_1 after R_2 departs. If the content needed by R_3 was missing from R_1's buffer, then R_3 could start streaming from the server after R_2's departute. In Figure 4, upon R_2 departure, the content that R_3 needs is not available in R_1's buffer and, hence R_3 is forced to stream from the server. The content missing from R_1's buffer is denoted as H in Figure 4. Similar to the case for a new arrival, if the total download rate is strictly greater than the playback rate, R_3 can open two simultaneous streams—one from R_1 and the other from the

server. Once it has obtained the missing content H from the server it can terminate its stream from the server and continue to download from R_1. Unlike the case for a new arrival, as discussed in Section 2.3, the stream patching technique may work in this situation even when the total download rate is less than twice the playout rate.

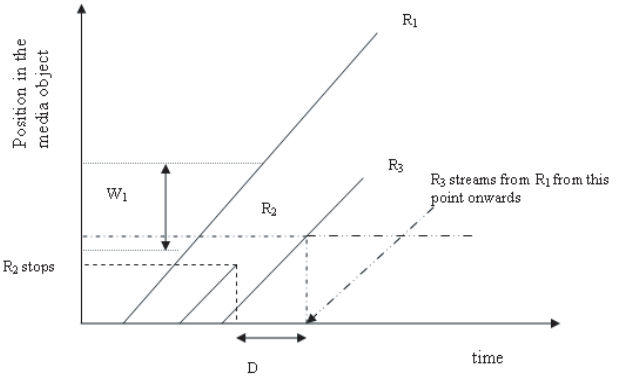

Fig. 5. Delay in finding a new download source

When the download rate is greater than the playout rate, a client can *prefetch* content to its buffer before it is time to playout that content. Prefetching 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 Figure 5. R_3 starts streaming from R_2 on arrival. After R_2 departs, it takes R_3 D seconds (time units) to discover the new source of download R_1. If the prefetched "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 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 prefetched content, until it discovers R_1. In a *cache-and-relay* strategy, clients do not prefetch content[1]. 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 (until 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

[1] It can be due to the fact that the playout rate is equal to the download rate or clients may choose not to prefetch content.

on R_2's departure in the case of *cache-and-relay*. In case of *prefetch-and-relay*, if the time required to playout the prefetched 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. Prefetching 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 prefetched content in its buffer.

2.1 Control Parameters

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

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. Given a particular α and T_b, it is easy to calculate that a client needs to download at the (total) rate α for $\left(\frac{\beta\alpha T_b}{(1+\beta)(\alpha-1)}\right)$ seconds to achieve the desired ratio β of "future" to "past" content in its buffer.

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.

2.2 Constraints in the Case of an Arrival

The following theorem is stated from [20]:
Theorem: A newly arrived client R_2 can download from the buffer of R_1 if the following conditions are satisfied:

- The inter-arrival time between R_1 and R_2 is less than T_b, or
- If the inter-arrival time between R_1 and R_2 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 ensures that the content needed by R_2 is present in R_1's buffer. The second condition defines the scenario in which the stream patching technique can be used by R_2 to "catch-up" with R_1.

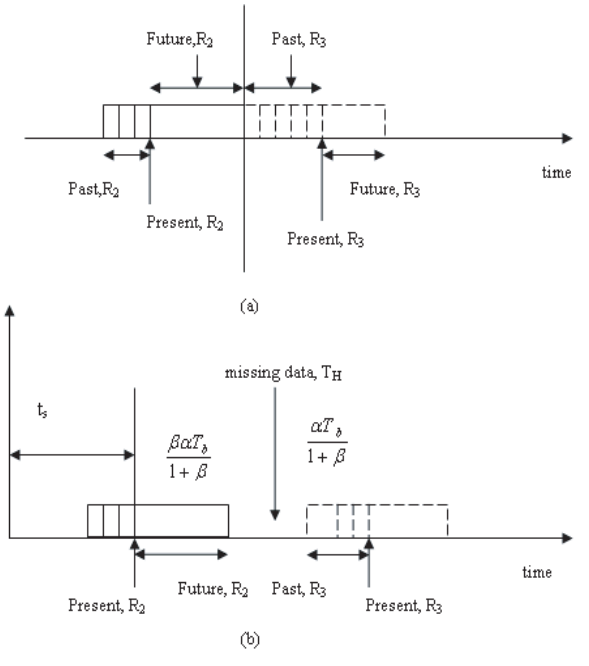

Fig. 6. Buffers of R_2 and R_3

2.3 Constraints in the Event of a Departure

Let us assume that R_2 was streaming from R_1's buffer. R_1 leaves the peer-to-peer network at time $t = t_d$. If the available buffer size at R_2 is $\alpha \times T_b$ bytes and at $t = t_d$, the ratio of "future" content to "past" content in R_2's buffer is β, then R_2 has $\left(\frac{\beta\alpha T_b}{1+\beta}\right)$ bytes of "future" content and $\left(\frac{\alpha T_b}{1+\beta}\right)$ bytes of "past" content in its buffer. At a *Playout rate* of 1 byte/time unit, R_2 has $\left(\frac{\beta\alpha T_b}{1+\beta}\right)$ time units to find a new source to download from after R_1 departs. Figure 6 presents the

state of R_2's buffer after R_1 departs. The time axis represents the progression of the playout of the stream. The shaded portion labelled as "Past" refers to the $\left(\frac{\alpha T_b}{1+\beta}\right)$ bytes of "past" content and the portion labelled "Future" refers to the $\left(\frac{\beta\alpha T_b}{1+\beta}\right)$ bytes of "future" content in R_2's buffer. The arrow marking "Present" refers to the position in the media content that has been played out at R_2.

If $\alpha = 1$, then after R_1's departure, R_2 can download from another client R_3's buffer if and only if the contents in their buffers (partially) overlap. Figure 6(a) shows the situation when the buffers of R_2 and R_3 are contiguous. Any client that is ahead of R_3, in terms of playing out the stream, would have some content that R_2 needs to download missing from its buffer and hence, unsuitable for R_2 to download from. Figure 6(b) depicts such a situation.

Consider the situation depicted in Figure 6(b). Let us assume that the ratio of "future" content to "past" content in R_2's buffer is β and hence, it currently has $\left(\frac{\beta\alpha T_b}{1+\beta}\right)$ bytes of prefetched data. Assume that the "missing" content is T_H bytes and that the playout rate is 1 byte/second. If $\alpha > 1$, then as discussed earlier, R_2 can open two simultaneous streams, one from the server and the other from R_3, and terminate its stream from the server after it has downloaded the "missing" content and continue to download from R_3 thereafter. Note that for this stream patching technique to work, R_3 should be over-writing contents in its buffer at a rate 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_3 should be over-writing its buffer at the rate of 1 byte/second. If this is the case, then R_2 can download from R_3 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, stated from [20], must be satisfied by the size of the "missing" content, T_H bytes, for R_2 to able to stream from R_3's buffer:

1. **Constraint imposed due to α:**

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

   The above inequality demands that the time taken to playout the prefetched and the "missing" content should be no less than the time taken to download the "missing" content. Note that if $\alpha \geq 2$, then condition (1) is always satisfied. The stream patching technique can be used in the case of a departure even when $1 < \alpha < 2$ if a client has sufficient prefetched content.

2. **Constraint imposed by the size of the buffer:**

$$T_H \leq \frac{\alpha T_b}{1+\beta} \tag{2}$$

   The above constraint demands that the size of the missing content, T_H, cannot be greater than the size of the "past" content in R_2's buffer.

3 Performance Evaluation

In this section, we compare the performance of the *prefetch-and-relay* based protocol, dPAM [20], and *cache-and-relay* with respect to savings in server bandwidth. We refer to the protocols oStream [6] and OSMOSIS [15] by the generic term *cache-and-relay* because they correspond to the situation where $\alpha = 1$ (hence, a client cannot prefetch any content.)

3.1 Simulation Model

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. The time spent by a client downloading the stream is exponentially distributed with mean 1000 time units (for Figures 7, 8 and 9), or 100 time units (for Figure 10.) The total number of client arrivals during each simulation run is 1500. The parameter β is set to 100,000—so that almost all the content in a client's buffer is "future" content after it has been downloading the stream long enough. For the simulation results presented in Figures 7, 8, 9 and 10, we assume that a client is able to determine whether it should stream from the server or from the buffer of some other client without any delays both in the case of arrivals as well as in the event of a departure; i.e. we do not take delays like propagation delays and the delay involved in searching for a suitable client to download from into consideration. Results in the presence of such delays can be found in [20]. In calculating the server load, we do not consider the small-duration streams that a client opens from the server to obtain the "missing" content.

3.2 Summary of Observations

If the download rate is sufficiently high, $\alpha \geq 2$, the *prefetch-and-relay* scheme of dPAM has an advantage over the *cache-and-relay* scheme in reducing the server bandwidth when the resources available for overlay stream multicast are constrained, for example when the buffer size is small or when the request arrival rate is low. This 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 *prefetch-and-relay* 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 *prefetch-and-relay* in terms of server bandwidth even when the download rate is very high.

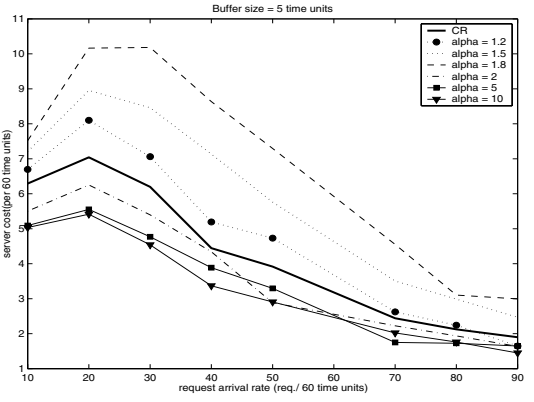

Fig. 7. Mean download time = 1000 time units, buffer size = 5 bytes

3.3 Simulation Results

For a fixed buffer size, we vary the download rate and measure the server load for different client arrival rates. Figures 7, 8 and 9 present the simulation results for different buffer sizes. For a fixed buffer size, *cache-and-relay* (*CR*) is able to match the performance of *prefetch-and-relay*, for all values of α considered in the simulation, once the client arrival rate increases beyond a certain threshold. This threshold depends on the size of the buffer — 80/(60 time units) in Figure 8 and 50/(60 time units) in Figure 9. The reason for this is two-fold. In these simulations, on average, a client downloads the stream for a duration (1000 time units) which is much longer than the time it needs to achieve the desired ratio β between its "future" and "past" content (approximately 5 time units for $\alpha = 5$ and buffer size of 20 bytes when β = 100,000.) Thus, most of the clients have achieved the ratio β by the time the client they were downloading from departs the peer-to-peer network. Since the value of β is set very high ($\beta = 100,000$), almost the entire buffer of a client is full of "future" content when the client it was downloading from leaves the overlay network. Hence, in the event of a departure, since condition (2) of Section 2.3 is often violated, clients are not able to take advantage of the higher download rate (by opening two simultaneous streams and doing stream patching.) Thus, on a departure, a client that needs a new source for download, can start downloading from another client only if their buffers (partially) overlap. This situation is similar to *cache-and-relay* ($\alpha = 1$) (refer to the discussion in Section 2) and hence, the stream patching technique employed in the dPAM protocol does not provide any advantage over *cache-and-relay*. If we set β to be small, say 2 or 3, then a client would be able to take advantage of the stream patching technique in the event of a departure.

The second reason for *cache-and-relay* being able to achieve the same server load as *prefetch-and-relay* beyond a certain request arrival rate threshold is as follows. A new arrival can take advantage of higher download rate and stream

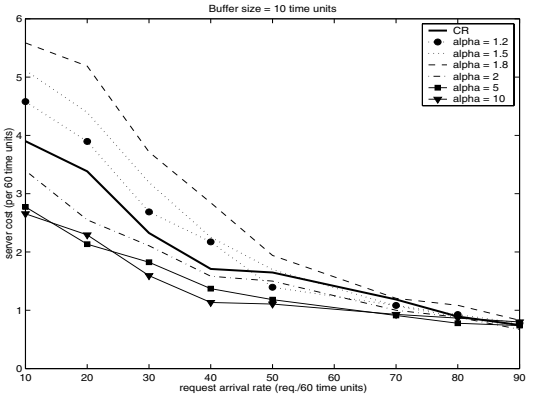

Fig. 8. Mean download time = 1000 time units, buffer size = 10 bytes

patching under dPAM but as the client arrival rate gets higher, a client rarely needs to resort to such capability. Hence, for a sufficiently high client arrival rate, the advantage that *prefetch-and-relay* has over *cache-and-relay*, due to clients being able to patch streams from the server (using a higher total download rate), is mitigated by the presence of a large number of clients in the peer-to-peer network enabling *cache-and-relay* to match the performance of *prefetch-and-relay*.

When $1 < \alpha < 2$, *prefetch-and-relay* leads to a greater server load than *cache-and-relay* for low arrival rates. As α increases, the time taken to fill the buffer at download speed, T_b, decreases. For example, for a buffer size of 5 bytes, 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[2]. 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 higher arrival rate.

When the download rate at least twice as fast as the playback rate ($\alpha \geq 2$) *prefetch-and-relay* achieves a much lower server load than *cache-and-relay* even for small buffer sizes and low request arrival rate. When $1 < \alpha < 2$, *prefetch-and-relay* does not have any advantage over *cache-and-relay*. Since in these simulations, we ignore the delay involved in searching for a suitable client to download from, the only advantage that *prefetch-and-relay* has over *cache-and-relay* comes from the fact that it enables a client to "catch-up" with another client by downloading the "missing" data from the server in the event of a departure. But for small α ($1 < \alpha < 2$), condition (1) of Section 2.3 is often violated. For example, with $\alpha = 1.2$ bytes/second, it will take 5 seconds to

[2] Since $1 < \alpha < 2$, a new arrival cannot take advantage of the stream-patching technique.

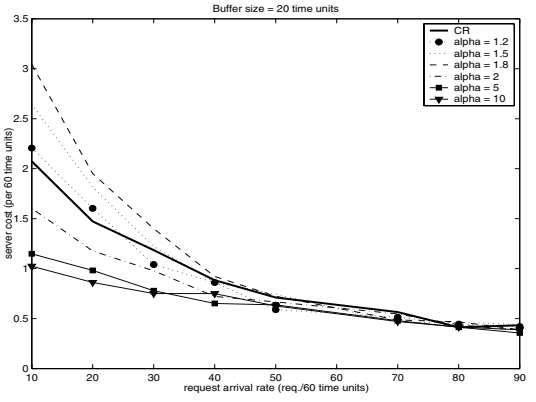

Fig. 9. Mean download time = 1000 time units, buffer size = 20 bytes

acquire a "missing" data of size 1 byte from the server. Hence, when the available buffer size is 5 bytes, a client never opens two simultaneous streams to do stream patching, because if it contains the required 5 bytes of "future" content to satisfy condition (1) of Section 2.3, then its buffer is already full of "future" content and it has no available buffer space to download the "missing" content from the server, which violates condition (2) of Section 2.3. On the other hand, if the client has less than 5 bytes of "future" content then condition (1) of Section 2.3 is violated. Combined with our observation in the preceeding paragraph, it becomes clear why *prefetch-and-relay* has a much higher server load compared to *cache-and-relay* for $1 < \alpha < 2$ when the client arrival rate is low.

The results in Figures 7, 8 and 9 show that as the available buffer at the client increases, the required server bandwidth to support a particular request arrival rate decreases, under both *cache-and-relay* as well as *prefetch-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 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. Figure 10 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. Figure 10 illustrates the fact that the *prefetch-and-relay* scheme with $\alpha \geq 2$ performs better than the *cache-and-relay* scheme, in terms of the server bandwidth re-

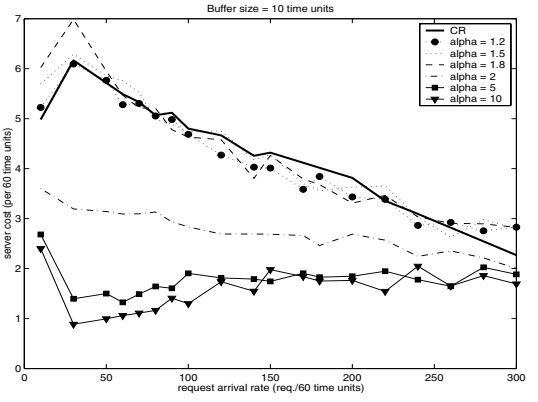

Fig. 10. Mean download time = 100 time units, buffer size = 10 bytes

quirement, even for very high client arrival rates (compare with Figures 7, 8 and 9.) Due to the shorter content download time of the clients, in a significant number of situations, clients are able to take advantage of the higher download rate through stream patching [12] in the event of a departure and hence, the server load is less under the *prefetch-and-relay* scheme compared to the *cache-and-relay* scheme. Figure 10 also shows that if the client request arrival rate keeps on increasing, eventually the *cache-and-relay* scheme will be able to match the performance of the *prefetch-and-relay* scheme.

4 Related Work

Delivery of streams to asynchronous clients has been the focus of many studies, including periodic broadcasting [22,13,11,17] and stream patching/merging techniques [4,10,7,8]. 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 [12] 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 caching has been proposed in several previous work [21,19,14,16]. The authors of [6] propose an overlay, multicast strategy, *oStream*, that leverages client-side caching to reduce the server bandwidth as well as the network link cost. Assuming the client arrivals to be Poisson distributed, they 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. *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 [15] 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 [1] using IDA [18], and more recently using the Digital Fountain approach which relies on Tornado encoding [3,2]. 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.

5 Conclusion

Through extensive simulations, we evaluated the performance of the distributed *prefetching* protocol, dPAM [20], proposed for scalable, asynchronous multicast in P2P systems. Our results show that when the download rate is at least twice as fast as the playout rate, a significant reduction in server bandwidth can be achieved, compared to a *cache-and-relay* strategy, when the resources available for overlay multicast are constrained, i.e. small client buffers and low client arrival rate. We evaluated the impact of departure of client nodes on the server bandwidth requirement, and highlighted the fact that the time spent by the clients downloading the stream is a crucial factor affecting the scalability of end-system multicast built upon client-side caching. We also discussed the advantage of prefetching content in improving the playout quality at the client nodes in the presence of various delays. We refer the reader to [20] for an extended analysis of dPAM in the presence of delays as well as its implementation.

References

1. A. Bestavros. AIDA-based Real-time Fault-tolerant Broadcast Disks. In *Proceedings of IEEE RTAS'96*, May 1996.
2. J. Byers, J. Considine, M. Mitzenmacher, and S. Rost. Informed content delivery across adaptive overlay networks. In *Proceedings of ACM SIGCOMM*, 2002.
3. 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.
4. 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.
5. Y. Chu, S. Rao, S. Seshan, and H. Zhang. Enabling Conferencing Applications on the Internet using an Overlay Multicast Architecture. In *Proceedings of ACM SIGCOMM*, 2001.

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. 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.
11. A. Hu. Video-on-demand Broadcasting Protocols: A Comprehensive Study. In *Proceedings of IEEE INFOCOM*, April 2001.
12. K. Hua and Y. Cai amd S.Sheu. Patching: A Multicast Technique for True On-demand Services. In *Proceedings of ACM Multimedia*, 1998.
13. 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.
14. 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.
15. 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.
16. D. Ma and G. Alonso. Distributed Client Caching for Multimedia Data. In *Proceedings of the 3rd International Workshop on Multimedia Information Systems(MIS97)*, 1997.
17. 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.
18. 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.
19. S. Ramesh, I. Rhee, and K. Guo. Multicast with Cache (mcache): An Adaptive Zero-delay Video-on-demand. In *Proceedings of IEEE INFOCOM*, 2001.
20. A. Sharma, A. Bestavros, and I. Matta. dPAM: A Distributed Prefetching Protocol for Scalable, Asynchronous Multicast in P2P Systems. In *Technical Report BUCS-TR-2004-026*, 2004.
21. 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.
22. 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.