On-Demand Multicast Streaming Using Collaborative Prefix Caching*

John Paul O'Neill and Jonathan Dukes

School of Computer Science and Statistics
Trinity College Dublin, Ireland
{jponeill,jdukes}@cs.tcd.ie

Abstract. Increasing mass-market acceptance of on-demand streaming services motivates us to seek new innovations in the way we deliver media content over networks. An architecture is proposed in which edge-resources in a peer-to-peer network assist in the provision of fully-interactive on-demand streaming services based on multicast. This approach represents a synergy between multicast batching, proxy prefix caching and collaborative caching of media content. The approach differs from other work in its use of long-term caching of content across streaming sessions. The proposed approach has been evaluated using a highly detailed network simulation that models real-world network (IPv6) and streaming (RTSP) protocols and the Pastry overlay network. Results demonstrate that substantial reductions in server bandwidth can be achieved with low client storage and bandwidth overhead.

1 Introduction

On-demand, Internet-based media streaming services are gaining mass-market acceptance as an alternative to traditional broadcast services for television and radio. The emergence and popularity of services such as BBC's iPlayer [1] and Hulu [2] is evidence of this. The strain that the popularity of these services places on network infrastructure has also been reported [3]. These factors motivate continued effort to seek new innovations in the delivery of media content.

This paper describes an approach to the delivery of on-demand media streams, in which edge-resources in a peer-to-peer network assist in the provision of on-demand streaming services based on multicast. The approach represents a synergy between a number of techniques, including multicast *Batching* [4], *Proxy Prefix Caching* [5] and collaborative caching [6]. These techniques can be combined to varying degrees, according to the behaviour the content provider wishes to see in the network.

Recent renewed interest in multicast support for delivery of multimedia content [7] has prompted a re-examination of its use in on-demand streaming applications. Multicast streaming has the potential to offer significant savings in the

* This work was supported by the Irish Research Council for Science, Engineering and Technology. The authors would like to acknowledge the support of the Grid Ireland Operations Centre personnel.

resources consumed by the delivery of popular streams. The challenge, however, is to service multiple clients with a single multicast stream, while still providing each client with full, independent, interactive control over the streams it receives. In particular, clients should experience low-latency when requesting playback of a new stream or moving to a new offset in an existing stream. *Patching* [8] has been proposed in the past as a means of addressing this challenge.

The use of peer-to-peer networks in applications such as PROMISE [9], Split-Stream [10], and the Multimedia Distribution Service proposed by Zhang et al. [11] has also received much attention. Applications such as these are based solely on the use of overlay networks and do not rely on central servers that store and supply all of the content, thereby removing the likelihood of a bottleneck occurring at any point in the network.

In contrast with these approaches, the architecture proposed here uses multicast as the primary delivery mechanism between servers and clients. Clients collaborate in a peer-to-peer network to cache and serve *prefix* streams containing the beginning of recently accessed streams. Delivery of these prefix streams can begin immediately to achieve low stream start-up latency, while delivery of multicast streams can be delayed to allow multiple clients to be *batched* into a single multicast stream, thereby reducing server bandwidth utilisation.

A detailed simulation of the proposed architecture has been developed. This simulation models real-world network (IPv6), stream control (RTSP) and multicast control (MLD and PIM-SSM) protocols and the Pastry [12] peer-to-peer overlay network. The simulation has been used to evaluate the hybrid architecture and demonstrate its effectiveness in reducing bandwidth utilisation between servers and localized groups of collaborating clients (e.g. on a local area network or in a neighbourhood). The combination of multicast and peer-to-peer techniques can achieve this reduction while using only a small proportion of each client's storage and bandwidth resources. The approach also allows service providers to achieve a balance between the use of multicast and reliance on client resources.

2 Background

The simplest way of implementing an on-demand streaming service that gives clients interactive control over stream playback is to provide each client with a dedicated, independent stream. Server and network resource requirements may be reduced in a simple manner by grouping together two or more client requests arriving over a short period of time and servicing those requests using a single multicast stream, an approach that has been referred to as *Batching* [4]. This approach, however, results in significantly higher stream startup latencies, since client requests must be delayed until a number of requests can be batched together. Furthermore, servicing multiple clients with a single multicast stream will result in the loss of independent, interactive client control over stream playback. *Patching* [8] was proposed as a way of using multicast to reduce network and server resource requirements, while still achieving low stream startup latencies.

Alternatively peer-to-peer architectures such as Joost [13], PPLive [14] and PPStream [15] offer a decentralized means of distributing on-demand content. Huang et al [16] have demonstrated that these architectures are becoming increasingly popular and that peer-to-peer on demand streaming is as viable as BitTorrent style download.

The use of peer-to-peer overlay networks in conjunction with multicast has been proposed by Lee et al. [17] and Ho et al. [18]. The architecture proposed here, however, differs significantly in the way in which cached content is managed, located and used. Both of these existing architectures use a central server to locate buffered content and select peers to service new client requests, rather than the decentralised approach proposed here. Furthermore, the collaborative prefix caching approach that is proposed here caches content for long periods across multiple streaming sessions, thereby allowing more extensive use of peer resources.

In the proposed architecture, *patch* streams of a defined maximum length are supplied by peers, rather than by the server. As a result, parallels can be drawn between this scheme and the *Proxy Prefix Caching* scheme proposed by Sen et al. [5]. Rather than relying on dedicated infrastructure to cache and serve prefix streams, however, peer resources are used to implement a collaborative, decentralised cache.

The collaborative prefix cache builds on the *Squirrel* decentralised peer-to-peer web cache proposed by Iyer et al. [6]. Instead of caching and serving small web objects, however, peers in the proposed architecture provide streams of cached multimedia content. The changes that result from this difference are discussed in Section 3.3.

The main advantage of the hybrid architecture is that it provides a reduction in bandwidth utilisation for the server, without causing clients to become fully dependant on peer-to-peer resources. By using peer resources at the edges of the network to supply cached prefix streams, the server is given an opportunity to delay the provision of streams, and thus group together numerous requests for the same content item and serve these requests with a single multicast stream.

3 Peer-Assisted Multicast

The challenge when implementing a multicast streaming service is to maximise the sharing of multicast streams while retaining low stream start-up latencies. The *Patching* scheme described in Section 2 achieved this by using a centralized server to provide both the multicast *batch* streams and the *patch* streams necessary to provide low-latency and client interactivity. A different approach is taken here, illustrated in Figure 1, in which clients collaborate to provide each other with the prefix streams necessary to facilitate low stream start-up latencies, leaving centralized servers to implement what effectively becomes multicast *Batching*.

Like Squirrel, it is assumed that collaborating clients are close together. Clients cache the prefix of streams that are required locally. When a client requests a stream, it first determines whether the prefix is available in the local

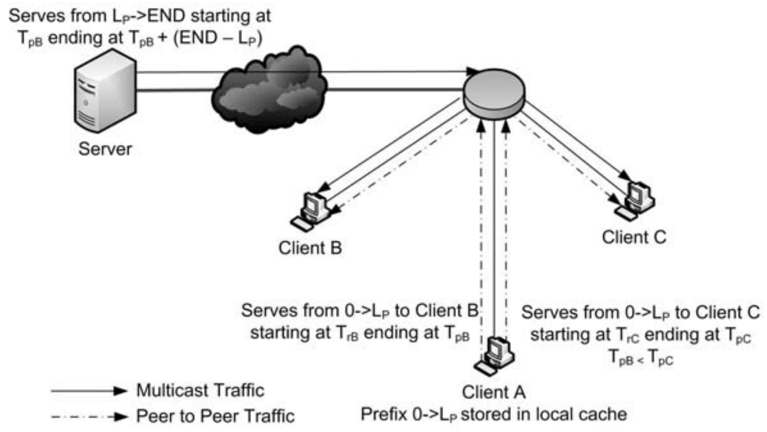
Serves from L_P->END starting at T_pB ending at T_pB + (END – L_P)
Server
Client B
Client C
Serves from 0->L_P to Client B starting at T_rB ending at T_pB
Serves from 0->L_P to Client C starting at T_rC ending at T_pC T_pB < T_pC
Multicast Traffic
Peer to Peer Traffic
Client A
Prefix 0->L_P stored in local cache

Fig. 1. Peer-Assisted Multicast. Clients B and C retrieve cached prefixes from Client A and the remainder of the stream from the server. The server needs to start providing the remainder by the time Client B has finished receiving the prefix.

cache, in which case only the remainder of the stream must be retrieved from the server. Otherwise, the client determines whether the prefix is available in the collaborative peer cache, in which case it retrieves the prefix from a peer and the remainder of the stream from the server. If the prefix is not available from either the local cache or the collaborative peer cache, the entire stream is retrieved from the server.

This approach will have the greatest effect on the server and network resources required to provide the most popular content streams, as the potential for multicast sharing for these streams is highest. The benefit for resource utilisation resulting from serving unpopular content will be significantly smaller since the server only benefits by not having to supply stream prefixes that clients can obtain from local or peer caches. Increasing or decreasing the prefix length will result in a server obtaining greater or lesser benefit respectively from the use of multicast.

3.1 Server Behaviour

A server can expect two types of request: those that must be served immediately and those that can be served at a future time. Requests that must be served immediately are those requests from clients who were unable to locate the prefix in either their local cache, or in the collaborative peer cache. As these requests must be served immediately, it is unlikely the server will be able to batch a number of these together, so the possibility of multicast sharing is minimal.

The second type of request is generated from clients who have located the prefix in either their local cache or a peer's cache. While these clients are viewing the prefix, the remainder of the content does not need to be served until the prefix is finished (allowing for some network delay). This window, between the

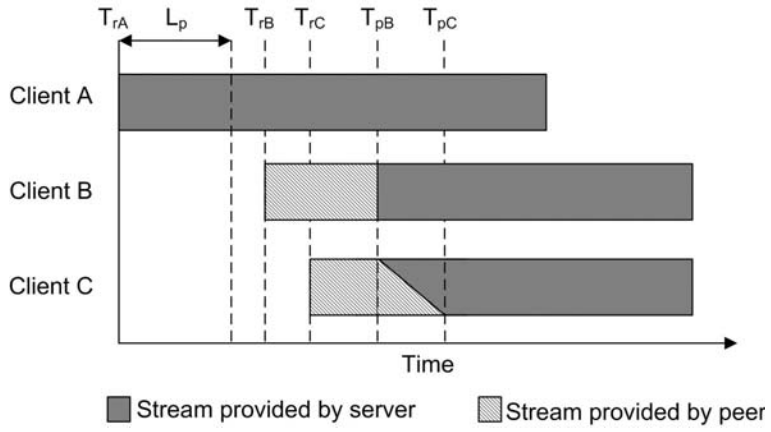

Fig. 2. Clients B and C retrieve cached prefixes from Client A and the remainder of the stream from the server

request arriving at the server and the required start time as specified by the client, allows the server to batch together other requests for the same content that arrive during this window.

Figure 2 illustrates the behaviour of the protocol. In this example, Clients A, B and C request the same stream. Client A was unable to locate the prefix in either its local cache or the collaborative cache so the client sends a request to the server at time T_{rA} which requires immediate playback of content. Once Client A has received the stream prefix of length L_p, it can supply that prefix to other clients. At time $T_{rB} > (T_{rA} + L_p)$, Client B issues a request for the stream. As the prefix is now stored in the collaborative cache, Client B requests the prefix from Client A and informs the server that it requires the content from L_p until the end of the stream, and that it must begin receiving the stream by T_{pB}. At time $T_{rC} < T_{pB}$, Client C sends a request identical to the request sent by Client B, except that in this case the stream needs to be served by T_{pC}. As there is already a request pending for this content, the server can satisfy both T_{pB} and T_{pC} by serving a multicast stream at $MIN(T_{pB}, T_{pC})$. Client C will need to buffer the stream from the server, and begin playback from the buffer at time T_{pC}.

3.2 RTSP Implementation

The simulation of peer-assisted multicast, described in Section 4, models the use of the RTSP [19] protocol for control of all multimedia streams in the system. For streams served by one peer to another using a unicast transport mechanism, the application of RTSP is straight forward. For streams served by a server using a multicast transport mechanism, however, the use of RTSP must be considered in more detail. A description of the use of RTSP to implement multicast *Patching* has been described previously by the authors [20]. Here, the focus will be on the most salient aspects of the use of RTSP to implement peer-assisted multicast. (It is assumed that an appropriate transport protocol, such as RTP, will be used for the delivery of unicast and multicast streams to clients.)

RTSP requires that the client and server participating in a session agree on the transport parameters including, in particular, the multicast address to which the client must subscribe, during the SETUP phase of the session. The server, however, must assign the client to a multicast batch without knowing when the client will issue a PLAY request, what range the client will request or what time the client expects playback to begin.

When the server receives a PLAY request, it is possible that the multicast batch to which it originally assigned the client is no longer suitable. To determine whether this is the case, the server checks whether the first packet in the range requested by the client has already been sent to the multicast address to which the client has subscribed. If this is the case, the server responds using the RTSP "Request Time-out" response and the client renegotiates the transport parameters with a new SETUP request.

In the case where the first packet in the range requested by the client has not yet been sent to the multicast address, the server then compares the playback start time requested by the client with the scheduled start of the current batch. If the scheduled start time is later, the server brings it forward to satisfy the start time requested by the client.

3.3 Collaborative Prefix Caching

Like the Squirrel decentralised web cache [6], clients in the proposed collaborative prefix cache collaborate in a peer-to-peer network to implement a cache of recently accessed content. Also like Squirrel, the Pastry [12] overlay network has been chosen as the substrate for the collaborative cache. This approach exploits temporal locality in the same way as a centralised cache but avoids the need for additional infrastructure. The differences between the collaborative prefix cache and the original Squirrel approach arise from the need to store and supply potentially lengthy, high-bandwidth media streams.

Both Squirrel and the collaborative prefix cache described here use key-based routing in a peer-to-peer network to locate content items. The authors of Squirrel explored two alternative approaches, which they termed *home-store* and *directory*. In both approaches, a request for a content item is routed through the peer-to-peer network to the *home node* whose identifier is numerically closest to the content item identifier.

In the *home-store* approach, each node is responsible for caching content items for which it is the home node. Thus, the cache at each node will contain both content items that were required locally and content items that were requested by other nodes and for which the node is the home node. Using the *directory* approach, home nodes are responsible for maintaining a directory of the nodes (as in Squirrel, these are termed *delegates*) that have a cached copy of the content items whose identifiers are closest to the node. Thus, each node only caches

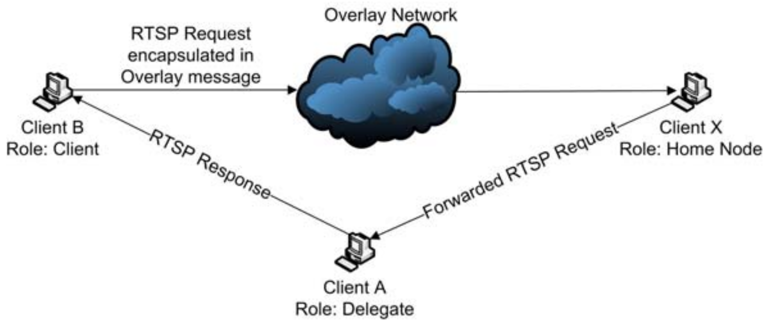

Fig. 3. Collaborative Cache Lookup

content items that were required locally. Figure 3 illustrates the roles played by peers in the location and supply of cached stream prefixes.

In their original work on Squirrel, the authors established that *home-store* performed better than *directory* with respect to its ability to both reduce external bandwidth usage (resulting from cache misses) and also balance load internally among participating nodes. However, since caching and serving high bit-rate media streams requires a significantly higher proportion of a client's bandwidth and storage resources, it is unreasonable to expect a client to retrieve, store and serve content that it does not itself require. Accordingly, a *directory* approach similar to that proposed for Squirrel is more appropriate for a collaborative prefix cache for media streaming applications.

The initial SETUP request issued by a client is treated in a similar manner to a HTTP GET request in Squirrel. This request may be conditional or unconditional, depending on whether the client has a fresh, cached copy of the requested stream prefix. The request is initially directed to the home node for the stream prefix, based on an identifier derived from the request URI.

The unconditional case is examined first. If the home node's directory contains one or more delegates with a cached copy of the requested content, the request is forwarded to a selected delegate. If the delegate's cached prefix is fresh, it will send an RTSP OK response to the client, establishing a new session. If the cached prefix is stale, the delegate will issue a conditional RTSP SETUP request to the server, the response to which will indicate whether the cached prefix has been modified. If the prefix has not been modified, the delegate will send an RTSP OK response to the client. If, however, the prefix is out of date, a new copy of the prefix must be retrieved from the server. The collaborative prefix cache differs from Squirrel at this point. In Squirrel, a delegate whose cached prefix has been modified will receive an up-to-date copy of the object from the server, in response to the conditional GET request. Retrieving a media stream that is not required locally, however, is likely to be a prohibitive use of client resources. Accordingly, the delegate issues an RTSP NOT FOUND response to the client and informs the directory that the prefix has been modified, causing it to be removed from the directory. The client will retrieve the entire stream from the server, caching the new prefix and eventually becoming the sole new delegate for the stream.

The case is now examined where a client has a locally cached prefix but the cached copy is stale. Again, the collaborative prefix cache behaves in a similar manner to Squirrel. The client issues a conditional RTSP SETUP request to the home node. Like Squirrel, the collaborative prefix cache maintains time-to-live information for each cached prefix. If the directory entry is fresh and the client's prefix has not been modified (we assume that this can be established through the use of Last-Modified timestamps) then the directory issues an RTSP NOT MODIFIED response to the client. If the prefix has been modified, then the request is forwarded to a chosen delegate and handled as described above. If the directory entry is not fresh, the client is instructed to contact the server to either validate its existing cached prefix or retrieve the entire stream from the server, thereby caching a new prefix. In either case, the client will update the directory with new information, allowing the directory to either validate all of the existing delegates or replace them with the requesting client as the new sole delegate.

When a client receives an RTSP OK response from a peer in response to a SETUP request, both it and the chosen delegate become participants in a new RTSP session. Although it would be possible at this point for the client and delegate to continue to communicate using the peer-to-peer network, a scheme in which the participants transfer to a more efficient, direct TCP connection is proposed. This has the effect of limiting the use of the peer-to-peer network to the location of home nodes and the occasional exchange of cached prefix metadata between delegates and home nodes.

Two approaches were considered to determine when clients advertise the availability of cached prefixes. In the first approach, clients wait until the entire prefix is cached before advertising the availability of the prefix to the directory node. Using this approach, the maximum delay until the server must begin streaming the remainder of the content is determined by the fixed prefix length. This in turn allows the server more time to batch requests together. The main disadvantage of this approach is that when the popularity of a content item increases suddenly (commonly referred to as a "flash crowd") there will be period of time during which peers are unable to provide prefix streams and hence the server will not be able to batch together requests.

Using an alternative approach, clients advertise the availability of a prefix as soon as they begin caching the stream and later update the directory when the entire prefix has been cached. If a client with an incomplete prefix is required to provide a prefix stream, it only provides the portion of the prefix that is currently cached. The receiving client must then obtain the remainder of the stream (from the end of the incomplete prefix to the end of the stream) from the server. Using this approach, the server has less time to group requests into a multicast stream. However, the approach also allows a larger pool of prefixes to be used when flash crowds occur.

Using both approaches, the prefix length represents the maximum any client will need to retrieve from the peer-to-peer network. This way clients will be rewarded with a more reliable stream from the server if they are viewing content for a sustained duration, instead of depending on content retrieved from the peer-to-peer network. If clients only view a short portion of streams, they will only be served by a peer stream.

Fairness is a consideration in any system that relies on the resources of peers to provide a service. To ensure fairness, we have chosen to limit each client to provide a single stream to another client at any time. If a client which is already providing a prefix is contacted by the directory node, then the client informs the directory that it has insufficient bandwidth and the directory contacts a different client to supply the stream. If all clients are providing content then the requesting client will need to retrieve the full stream from the server.

The use of a round robin delegate selection policy when providing complete prefixes will ensure fairness. For incomplete prefixes, a round robin selection policy would ensure fairness but could result in the use of short prefix streams. The longer the prefix a delegate can provide, the more beneficial it is to the system as a whole as servers are given more time to delay the start of multicast streams. For maximum benefit to the server, incomplete prefixes are only used if there are no complete prefixes available.

4 Performance Evaluation

The performance of the proposed hybrid architecture has been evaluated using the OMNeT++ [21] discrete event network simulation system. Real-world network protocols have been modelled to allow an exploration of the implementation of the architecture, in addition to its performance. An IPv6 network has been modelled, using the MLD [22] and PIM-SSM [23] protocols for multicast control. The main parameters used are listed in Table 1. These parameters have been chosen to subject the server to a mean load of 900 concurrent streams if no multicast is used. To provide a continous flow of requests a client population greater than this needs to be maintained. Although a cache TTL is used, it was assumed for these experiments that the content is never modified by the server. A Zipf distribution has been used to model the popularity of content items. The total duration of each simulation is 48 hours and the first six hours of each simulation have been ignored to allow the system to stabilize.

To investigate the sudden introduction of new, popular content to the system, akin to a "flash crowd", ten new items were added at regular intervals of six hours, with the first new items introduced eighteen hours into the simulation. These new items became the most popular items according to the Zipf distribution and the original content items were all reduced in popularity by ten places.

Figure 4 illustrates the effect of prefix length on the mean number of streams served concurrently by the server and by each client for the complete prefix selection scheme. The schemes which used incomplete prefixes produced similar results, which are omitted for brevity. Prefix length has an approximately linear relationship with the mean number of streams served by each client. The effect

Table 1. Experiment Parameters

Parameter	**Range**
Number of Content Items	5000
Mean Stream Length (seconds)	3600
Prefix Length (seconds)	0-1800
Cache Size (number of prefixes)	5
Network Size (number of clients)	1000
Request Rate (requests/second)	0.25
Zipf Skew Factor	1.0

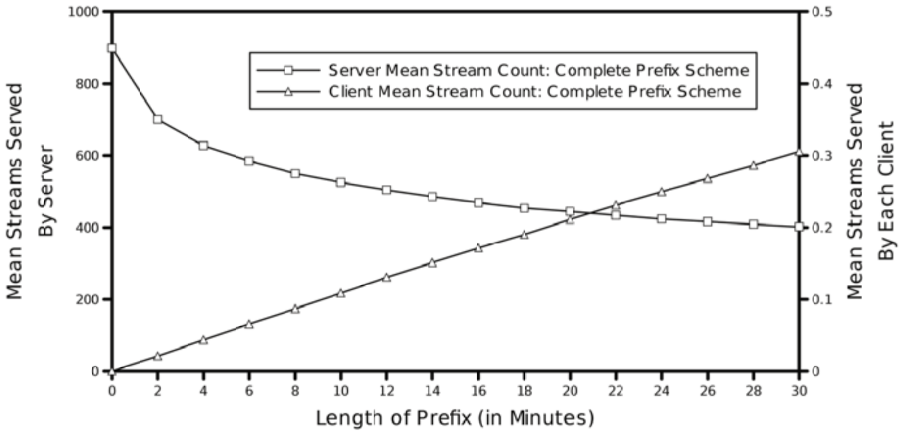

Fig. 4. Mean number of concurrent streams served by the server (left axis) and by any client (right axis)

of prefix length on the mean number of streams served by the server initially shows a sizable reduction for even a small increase in prefix length. As the prefix length increases, the server is able to serve more clients with each multicast stream, thereby reducing the number of multicast streams required. Further increases in prefix length have a diminishing effect.

The effect that varying prefix length has on the number of clients served by each multicast stream is linked to the popularity of the content. This can be observed in Figure 5, which shows that the majority of the benefit is seen for only the most popular content items, but that for these items, even relatively small prefix lengths result in a significant reduction in server bandwidth. (e.g. A six minute prefix allows the server to service a mean of ten clients with one multicast stream for the most popular content item.)

The effect of a flash crowd when using a prefix length of thirty minutes can be seen in Figure 6. If only complete prefixes are used, the server must deliver significantly more streams than the mean (solid horizontal line) for a sustained period of time. If a round robin scheme is used to select incomplete prefixes,

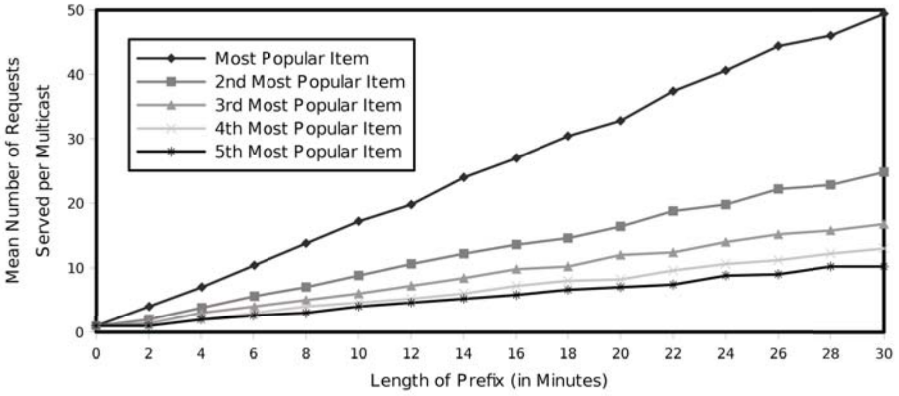

Fig. 5. Mean number of requests serviced per multicast stream for the five most popular content items

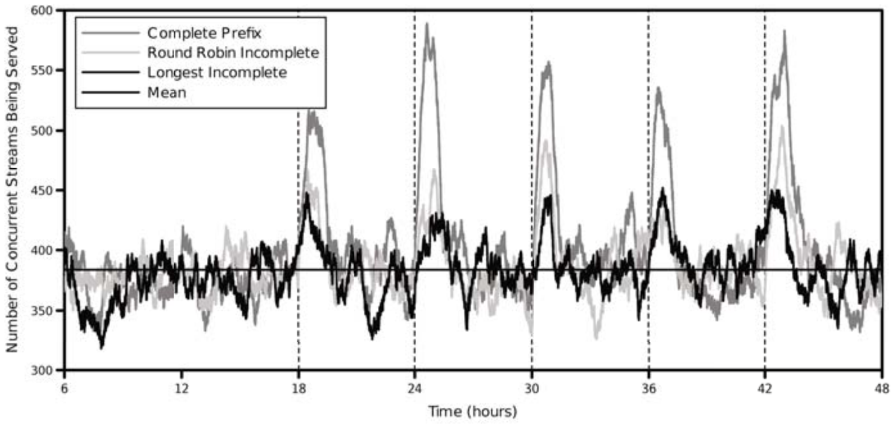

Fig. 6. Effect of flash crowds on the system. The timing of each flash crowd is illustrated with a dotted vertical line.

however, server load is reduced. If a scheme that selects the longest available incomplete prefix is used, then the effect of flash crowds is reduced further.

The effect on the main server of using incomplete prefixes is also illustrated in Figure 7, which shows the server behaviour around the peaks seen in Figure 6. The time at which the server is required to start each new stream to serve a batch of requests for one of the newly introduced content items is plotted against the cumulative number of such streams served. As can be seen, using complete prefixes requires the most time for the server to stabilise, since the server cannot batch together requests until a sufficient number of prefixes are available. Using the longest available prefix, however, allows the maximum time for requests to be batched together and, as a result, the server requires less time to stabilise.

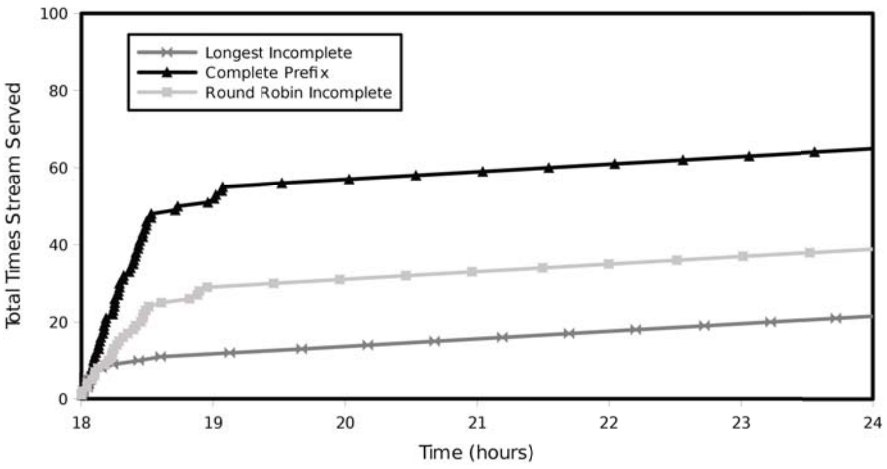

Fig. 7. Effect of new popular content

The rate at which the server provides stream of these popular content items eventually stabilises using all three policies, once there is a large enough pool of complete prefixes available in the collaborative cache.

5 Discussion and Conclusions

The use of peer-assisted multicast batching can significantly reduce server and network bandwidth utilisation by allowing servers to take advantage of the benefits of multicast streaming, without incurring high playback latencies or sacrificing independent client control over playback. To provide servers with the opportunity to batch multiple clients into a single multicast stream, clients implement a collaborative prefix cache containing the start of recently accessed streams. A peer-to-peer overlay network can be used to facilitate the location of cached content within the collaborative cache. This flexible approach gives service providers the ability to balance the benefits gained from multicast with reliance on client resources at the edges of the network. The work described in this paper differs from other proposals to combine multicast with peer-to-peer overlay networks in its use of a long-term, collaborative prefix cache.

A detailed simulation study of the use of collaborative prefix caching has been conducted, which allowed an investigation of its practical implementation in addition to its performance. The simulation model is based on the use of real-world network (IPv6), streaming (RTSP) and multicast control (MLD and PIM-SSM) protocols and on the use of the Pastry overlay network.

The results from the performance evaluation show that peer-assisted multicast can significantly reduce the server and network resources required to serve popular content items, while using only a small proportion of each client's storage and bandwidth resources. For example, using a prefix length of ten minutes yielded a reduction in the average number of streams served by the server of approximately 40%, while requiring an average of 9% of the client population to serve prefix streams concurrently. The results also show that a high degree of load balancing can be achieved with even a simple delegate selection scheme such as round-robin.

The effect of "flash crowds", which occur when there is a rapid increase in the demand for one or more content items, has been demonstrated by simulating the introduction of new popular content. Flash crowds have the largest effect when only complete prefixes are used, whereas a system based around using the longest available incomplete prefix suffers least from a flash crowd.

As with any system based on peer-to-peer technology, the effect of churn, which occurs when peers join and leave the network, must be a consideration as high levels of churn can destabilize a peer-to-peer network. Also, as this system allows peers to renege on requests if they are already providing a stream, it is open to abuse by a malicious peer not wishing to contribute resources to the system. This behaviour is commonly referred to as free loading. The ability of the hybrid architecture to handle churn and freeloading will be the subject of future research. The detailed network simulation will also allow an evaluation the effect of network location on the performance of peer-assisted multicast batching.

References

1. BBC iPlayer, `http://www.bbc.co.uk/iplayer/` (Last accessed July 13, 2009)
2. Hulu: `http://www.hulu.com/` (Last accessed July 13, 2009)
3. Wakefield, J.: BBC and ISPs clash over iPlayer (April 2008), `http://news.bbc.co.uk/1/hi/technology/7336940.stm` (Last accessed July 13, 2009)
4. Dan, A., Sitaram, D., Shahabuddin, P.: Scheduling policies for an on-demand video server with batching. In: Proceedings of the 2nd ACM International Multimedia Conference, San Francisco, California, USA, October 1994, pp. 15–23 (1994)
5. Sen, S., Rexford, J., Towsley, D.: Proxy prefix caching for multimedia streams. In: Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 1999), New York, USA, March 1999, pp. 1310–1319 (1999)
6. Iyer, S., Rowstron, A., Druschel, P.: SQUIRREL: A decentralized, peer-to-peer web cache. In: Proceedings of the 12th ACM Symposium on Principles of Distributed Computing (PODC 2002), Monterey, California, USA (July 2002)
7. BBC Multicast, `http://www.bbc.co.uk/multicast/` (Last accessed July 13, 2009)
8. Hua, K.A., Cai, Y., Sheu, S.: Patching: A multicast technique for true video-on-demand services. In: Proceedings of the 6th ACM International Multimedia Conference, Bristol, UK, September 1998, pp. 191–200 (1998)
9. Hefeeda, M., Habib, A., Botev, B., Xu, D., Bhargava, B.: PROMISE: peer-to-peer media streaming using collectcast. In: MULTIMEDIA 2003: Proceedings of the 11th ACM International Conference on Multimedia, pp. 45–54. ACM, New York (2003)

10. Castro, M., Druschel, P., Kermarrec, A., Nandi, A., Rowstron, A., Singh, A.: Splitstream: High-bandwidth multicast in cooperative environments. In: Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP 2003), October 2003, pp. 298–313 (2003)
11. Qin Zhang, Y.: Peer-to-peer based multimedia distribution service. IEEE Transactions on Multimedia 6(2), 343–355 (2004)
12. Rowstron, A., Druschel, P.: Pastry: Scalable, decentralised object location and routing for large-scale peer-to-peer systems. In: Guerraoui, R. (ed.) Middleware 2001. LNCS, vol. 2218, pp. 329–350. Springer, Heidelberg (2001)
13. Joost: `http://www.joost.com/` (Last accessed July 13, 2009)
14. PPLive: `http://www.pplive.com/` (Last accessed July 13, 2009)
15. PPStream: `http://www.ppstream.com/` (Last accessed July 13, 2009)
16. Huang, Y., Fu, T.Z., Chiu, D.M., Lui, J.C., Huang, C.: Challenges, design and analysis of a large-scale P2P-VoD system. SIGCOMM Comput. Commun. Rev. 38(4), 375–388 (2008)
17. Lee, C.H., Gui, Y.Q., Jung, I.B., Choi, C.Y., Choi, H.K.: A peer to peer prefix patching scheme for vod servers. In: Proceedings of the 3rd International Conference on Information Technology: New Generations, Las Vegas, Nevada, USA (April 2006)
18. Ho, K., Poon, W., Lo, K.: Enhanced peer-to-peer batching policy for video-on-demand system. In: International Symposium on Communications and Information Technologies (ISCIT 2006), September 2006, pp. 148–151 (2006)
19. Schulzrinne, H., Rao, A., Lanphier, R.: Real time streaming protocol (RTSP). IETF RFC 2326 (April 1998)
20. O'Neill, J., Dukes, J.: Re-evaluating multicast streaming using large-scale network simulation. In: Proceedings of the First International Conference on Intensive Applications and Services (April 2009)
21. Varga, A.: The OMNeT++ discrete event simulation system. In: Proceedings of the European Simulation Multiconference (ESM 2001), Prague, Czech Republic (June 2001)
22. Vida, R., Costa, L.: Multicast listener discovery version 2 (MLDv2) for IPv6. IETF RFC 3810 (June 2004)
23. Bhattacharyya, S.: An overview of source-specific multicast (SSM). IETF RFC 3569 (July 2003)