PROP: a Scalable and Reliable P2P Assisted Proxy Streaming System [*]

Lei Guo, Songqing Chen, Shansi Ren, Xin Chen, and Song Jiang
Department of Computer Science
College of William and Mary
Williamsburg, VA 23187, USA
{lguo, sqchen, sren, xinchen, sjiang}@cs.wm.edu

Abstract

The demand of delivering streaming media content in the Internet has become increasingly high for scientific, educational, and commercial applications. Three representative technologies have been developed for this purpose, each of which has its merits and serious limitations. Infrastructure-based CDNs with dedicated network bandwidths and powerful media replicas can provide high quality streaming services but at a high cost. Server-based proxies are cost-effective but not scalable due to the limited proxy capacity and its centralized control. Client-based P2P networks are scalable but do not guarantee high quality streaming service due to the transient nature of peers. To address these limitations, we present a novel and efficient design of a scalable and reliable media proxy system supported by P2P networks. This system is called **PROP** *abbreviated from our technical theme of "collaborating and coordinating* **PRO***xy and its* **P***2P clients". Our objective is to address both scalability and reliability issues of streaming media delivery in a cost-effective way. In the PROP system, the clients' machines in an intranet are self-organized into a structured P2P system to provide a large media storage and to actively participate in the streaming media delivery, where the proxy is also embedded as an important member to ensure quality of streaming service. The coordination and collaboration in the system are efficiently conducted by our P2P management structure and replacement policies. We have comparatively evaluated our system by trace-driven simulations with synthetic workloads and with a real-life workload trace extracted from the media server logs in an enterprise network. The results show that our design significantly improves the quality of media streaming and the system scalability.*

1. Introduction

Delivering multimedia contents with high quality and low cost over the Internet is challenging due to the typical large sizes of media objects and the continuous streaming demand of clients. There are three representative solutions for media streaming in the Internet. First, special content delivery networks (CDN) have been built to replicate media servers across the Internet to move the contents close to the clients [1]. This approach is performance-effective but is also very expensive. The second approach is to utilize existing proxies to cache media data, which is cost-effective but not scalable due to limited storage and bandwidths of centralized servers. The third approach is to build client-based P2P overlay networks for media content delivery, which is highly cost-effective but does not guarantee the quality of services because the capacities of peers can be heterogeneous and their availabilities can be transient. Our proposed system attempts to address both the scalability and the reliability issues of streaming media delivery in a cost-effective way.

Many researchers delve in the media delivery problem by looking into the proxy caching approach, which has been successfully used for delivering text-based content on the Internet. However, a full caching approach of media objects can quickly exhaust the limited proxy cache space. To handle the large sizes of media objects, researchers have developed a number of segment-based proxy caching strategies (e.g.[2], [3]) to cache partial segments of media objects instead of their entirety.

Although the segment-based proxy caching technique has shown its effectiveness for media streaming, the quality of service it can provide is still not satisfactory to clients for the following reasons. First, the limited storage capacity of a proxy will restrict the amount of media data it can cache for clients. Second, the delivery of streaming media normally requires a dedicated reservation of continuous bandwidths for the clients, thus, the highly demanded proxy bandwidths will limit the number of clients to be served simultaneously. Furthermore, a proxy not only easily becomes a system bottleneck, but also forms a single point of failure, being vulnerable to attacks. On the other hand, the resources of bandwidth, storage, and CPU cycles are richly available and under-utilized among the clients. Thus, the P2P file sharing model is very attractive. However, directly borrowing this model for media streaming can not guarantee the quality of streaming service for the following three reasons. First, the availability of demanded media data in each peer can be

∗ This work is supported in part by the U.S. National Science Foundation under grants CCR-0098055 and ACI-0129883 and a grant from the HP labs.

unpredictable because each peer caches and replaces media content independently without any coordination with other peers. Second, the availability of services can also be dynamic due to the transient nature of peers even though the data is always available. Third, the quality of services provided by multiple collaborative peers sometimes may not be sufficient for highly dynamic and bursty media streaming requests.

In order to address the scalability problem of proxy-based techniques, and to deliver the media content with high quality to the clients, we present a novel and scalable segment-based P2P media delivering system by organizing the proxy and its clients in the same intranet into a P2P system. In such a system, the clients effectively coordinate and collaborate with the proxy to provide a scalable media storage and to actively participate in the streaming media delivery. Media objects are cached in segment unit both in peers and in the proxy for the purposes of both self-viewing and global sharing. We have comparatively evaluated our proposed system by trace-driven simulations with synthetic workloads and with a real-life workload trace extracted from the media server logs in an enterprise network. We have shown that our design significantly improves the quality of media streaming and the system scalability. Our contributions in this work are three-fold.

- The collaboration and coordination between the proxy and its P2P clients in our system address the scalability problem of proxy-based technique, and also eliminate the concern of unstable quality of services by only relying on self-organized clients. These two system components are complementary to each other: the proxy provides a dedicated storage and reliable streaming services when peers are not available or not capable to do so, while peers provide a scalable storage for data caching and significantly reduce the service load of the proxy.
- Our proposed content location mechanism in the system is efficient and cost-effective. The segment access information and the location of peers holding segments are simply abstracted into a distributed hash table across the P2P network. Locating content in our system consists of two steps: (1) the system locates the peer maintaining the index of the requested segment, and (2) the indexing peer selects a serving peer to provide the segment. Since the index in step (1) contains global information about the segment, the system is able to balance the workload and improve the service quality.
- The load balance and data locality in the PROP system are determined by the segment replacement policies. We have proposed two replacement policies, one for peers and one for the proxy. Our objective is to keep popular media segments in the proxy for global sharing, and leave a certain space in each peer to cache relatively unpopular segments. With this arrangement, both popular and unpopular segments are fairly treated, improving the overall hit ratios in the PROP system. Our proxy replacement policy is popularity-based, considering both the recent and past access information of media segments, maximizing the cache utilization of the proxy. Our peer replacement policy is utility-based, considering both the popularity and the number of copies of the segments already in the system, giving a fair distribution of media data in the system.

2. Other Related Work

Efforts have been made on P2P media streaming only recently. Authors in [4] and [5] propose P2P multicast trees for live media streaming. Although multicast can be used for on-demand media streaming as well, the startup latency is non-trivial. Authors in [6] and [7] propose P2P streaming schemes for layer-encoded media. The quality of layer-encoded media streaming may not be guaranteed if some layers are not delivered in time, which can not occur in segment-based system. In weakly coupled P2P systems such as the ones presented in [6] and [5], users collaborate loosely to each other in a peer-to-peer fashion for on-demand media streaming, instead of being organized into a P2P overlay. Authors in [8] and [9] propose a Gnutella-like unstructured P2P media streaming system and a structured P2P media streaming system respectively. Both systems target media sharing in the global Internet. In contrast, our system targets on-demand media streaming from professional and commercial media providers for clients in a large intranet.

3. Our System Design and Rationale

3.1. Infrastructure Overview

The two main components of the PROP system are (1) the proxy and (2) all the client peers receiving the media streaming service, which are connected by a P2P overlay network. The proxy is the bootstrap site of the P2P system and the interface between the P2P system and media servers. When an object is requested for the first time or when no peers in the system are able to serve a streaming request, the proxy is responsible for fetching the requested media data from remote server, caching them locally, and segmenting the object into small units evenly.

Each peer in the PROP system has three functionalities. First, a peer is a *client* that requests media data; second, a peer is a *streaming server* that provides media streaming service to clients. Each peer caches the media data in segments while its content accessing is in progress, and shares the cached data with other peers in the system. Third, a peer is also an *index server* that maintains a subset of indices of media segments in the system for content locating. Peers in our system are self-organized into a structured P2P overlay supporting a distributed hash table, which maps the identifier of each media segment to the index of the segment (see Section 3.2 in details). The P2P operations in our system are overlay independent. We use CAN [10] in our simulation, however, other P2P overlay structures, such as Chord [11], Pastry [12], and Tapestry [13], can also be used.

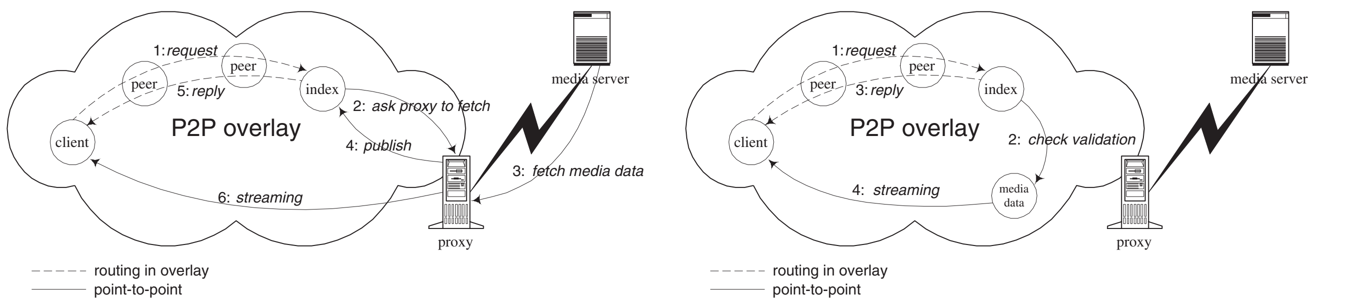

Figure 1. The requesting and serving of media data. Left: The proxy fetches and serves the requested media data; Right: A peer serves the requested media data.

In our system, the media segments and their corresponding indices are decoupled. In other words, they are maintained by different peers. The index of a segment contains a location list of peers, each of which caches a copy of the media segment, and the access information of this segment, which is used for replacement operations. The segment locating is conducted in two steps: the first step is to route the request to the peer maintaining the index of the demanded segment, and the second step is to select a peer that caches a copy of the segment. This approach is efficient and cost-effective for two reasons. First, the selection of a serving peer can be optimized according to the capacities and workloads of peers caching the demanded media data, because the index server maintains all access information of segments. Second, the cost of content locating is distributed over the P2P network so that the workload of P2P routing on each peer is trivial. Once the demanded segment is successfully located, the media streaming between the serving peer/proxy and the requesting peer becomes point-to-point.

3.2. P2P Routing and Media Streaming

The distributed hash table supported by the P2P overlay stores the $(key, value)$ maps where each key is the identifier of a media segment and the corresponding $value$ is the index of the segment. The identifier of a media segment is a GUID (globally unique identifier) hashed from the URL of the media object and the offset of the segment in the object. In our system, each peer is assigned a key space zone when joining the system, and maintains the segment indices mapped to this zone. Joining P2P routing entails getting the key space zone and take over the corresponding indices from a neighbor while leaving P2P routing entails handing over the segment indices and merge the key space zone to a neighbor [10].

The following operations on the distributed hash table are designed in our system for content locating and data management: $publish$, $unpublish$, $request$, $update$ and $notify$.

3.2.1. Publishing and Unpublishing Media Segments The $publish(seg\_id, location)$ operation publishes a cached copy of media segment in the P2P system, in which seg_id is the segment identifier, and $location$ is the IP address and port number of the peer that caches the segment copy. Correspondingly, the $unpublish(seg\_id, location)$ operation unpublishes the copy of media segment stored in $location$. To publish or unpublish a segment, a peer routes its $location$ and the seg_id to the target peer that maintains the segment index. Then the target peer put the $location$ into or remove it out of the location list in the segment index. A peer publishes a segment as soon as it caches the full segment and unpublishes a segment as soon as it deletes the segment. A peer publishes all segments it caches when joining the P2P system and unpublishes all segments it caches when leaving the P2P system.

3.2.2. Requesting and Serving Media Segments A peer requests media data segment by segment, and searches in its local cache first. If the local search fails, it calls the $request(seg\_id, URL)$ operation, which requests a segment of the object designated by the URL. When a peer requests a media object that it does not cache, it routes the URL to the target peer that maintains the key space zone that the identifier of the object's first segment is mapped to. If the corresponding index does not exist, meaning the object is requested for the first time, the target peer sends a request to the proxy, which fetches the requested object from the media server, and creates the index and publishes the object. Then the target peer routes the proxy's location back to the requesting peer, redirecting the peer to the proxy to get the media data. If the target peer finds the segment index, but the location list is empty, the target peer sends a request to the proxy, which fetches the segment and publishes it. The first five steps in Figure 1 (left) show such an example. If the location list is not empty, the target peer checks the validation of each location link, then returns the location of the peer with the maximal available bandwidth to the requesting peer. The first three steps in Figure 1 (right) show such an example. Then the serving peer provides the requested data to the requesting peer, as the last step shown in Figure 1 (left) and Figure 1 (right). A client buffers the next segment when the current segment is played back. If a serving peer wants to leave the P2P system before the current streaming terminates, it must push the rest of the segment to the requesting peer before exiting the P2P system.

3.2.3. Updating Segment Popularity and Utility Values PROP uses the popularity and utility values of segments to manage cached data (see Section 3.3 in details). These values depend on the access information and number of copies of corresponding media segments. When the proxy or a peer finishes serving a segment streaming task, it calls $update(seg\_id, access\_info)$ operation, which routes the access information to the target peer maintaining the segment's index, and then the target peer updates these information items. When the segment popularity or utility values change, the index server notifies all peers that cache the segment new values by $notify(peerset, seg\_id, value)$ operation, where $peerset$ is the peers in the location list of the segment index, and $value$ is the popularity or utility value of the segment designated by seg_id.

3.2.4. Message Routing Overhead In PROP, for each segment a client requests, a $request$ and an $update$ message are generated. For each segment replica that is cached or evicted from cache, a $publish$ or $unpublish$ message is generated. Although a $notify$ operation may generate multiple messages, it can be postponed if the popularity or utility value changes little. Further more, our replacement algorithm can keep the location list of the segment index in a moderate size (see Section 3.3 for details). Thus, the routing overhead in PROP is trivial compared to the media data transfered. Our simulation shows it is less than 1% of the streaming media data (including TCP/IP and Ethernet headers). To further reduce the routing overhead, we can increase the segment size, or use variable-sized segmentation such as exponential segmentation [2] and adaptive and lazy segmentation [3].

3.3. Global Replacement Policies

In our system, the proxy serves as a permanent cache site, but the storage size is limited. On the other hand, the total storage space contributed by peers is huge but the available contents change dynamically because peers come and go frequently. To fully utilize the storage and to improve the streaming service quality, we propose efficient replacement policies for both proxy and peers based on the global information of segment accesses using the following variables.

- T_0, the time when the segment is accessed for the first time;
- T_r, the most recent access time of the segment;
- S_{sum}, the cumulative bytes that the segment has been accessed;
- S_0, the size of the segment in bytes;
- n, the number of requests for this segment;
- r, the number of replicas of the segment in the system.

3.3.1. Popularity-based Proxy Replacement Policy The proxy takes over the streaming service whenever the requested media segment can not be served by any peer in the system. Thus, the proxy should hold those popular media objects to minimize the performance degradation due to peer failures. We use a popularity-based replacement policy instead of LRU policy, because LRU is not efficient for file scan operations, which are typical in media streaming services, and can only exploit the locality of the accesses to the proxy instead of the whole system. Similar to the cache utility function proposed in [3], which describes the popularity of objects in proxy caching systems, we define the popularity of a segment as

$$p = \frac{\frac{S_{sum}}{S_0}}{T_r - T_0} \times \min(1, \frac{\frac{T_r - T_0}{n}}{t - T_r}), \qquad (3.1)$$

where t is the current time instant, $\frac{\frac{S_{sum}}{S_0}}{T_r - T_0}$ represents the average access rate of the segment in the past, normalized by the segment size, and $\min(1, \frac{\frac{T_r - T_0}{n}}{t - T_r})$ represents the probability of future access: $\frac{T_r - T_0}{n}$ is the average time interval of accesses in the past, if $t - T_r > \frac{T_r - T_0}{n}$, the possibility that a new request arrives is small; otherwise, it is highly possible a request is coming soon. The segment with the smallest popularity is chosen as the victim to be replaced when the proxy cache is full. Considering both the recent access and past access information, the proxy can cache the most useful media data for clients.

3.3.2. Utility-based Peer Replacement Policy in Client Peers Independently exploiting access locality on each client side is neither efficient from the system's perspective nor effective from the user's perspective. First, due to the client access patterns, the popular objects get more accesses from peers and thus have more copies cached in the system, which are already cached on the proxy side. Keeping those unnecessary copies of popular objects degrades the cache efficiency since the cache space could have been used to cache other objects. Second, the locality on each client side is limited and the cached data is prone to be flushed in a long streaming session if LRU replacement is used. Third, the segments of a media object may be cached in a single peer, thus the data availability is very sensitive to the peer failure and leaving. On the other hand, the access locality of all clients is much more significant than that of a single client and the difference between the access latency in an intranet and the local disk is unimportant for media streaming. Thus, in PROP system, the cached data of all clients are maintained collectively in a decentralized fashion by replacement operations.

Our replacement policy in peers is designed to replace both those media segments with diminishing popularities because they rarely get accessed, and those popular media segments with too many copies being cached. As a result, peers accessing media objects completely will cache the latter segments and evict the beginning segments of the objects because they are more popular and have more replicas in the system than the latter segments. Peers that access only the beginning segments will cache the beginning segments. Thus, naturally a peer will cache only a few segments of each object it has accessed, while the segments of each object are distributed in many peers in the system according to their popularities, reducing the negative effects of peer failures.

For the peer replacement policy, we define the utility value of a segment as

$$u = \frac{(\log p - \log p_{min}) \times (\log p_{max} - \log p)}{r^{\alpha+\beta}}, \qquad (3.2)$$

where p represents the popularity of the segment, p_{min} and p_{max} estimate the minimum and maximum of segment popularities in the P2P system respectively, and r is the number of replicas of this segment in the system. The values of p_{min} and p_{max} can be maintained by the proxy and propagated across the P2P overlay by flooding when necessary. The term $\frac{\log p - \log p_{min}}{r^\alpha}$ captures the segments with small popularities and large numbers of replicas while $\frac{\log p_{max} - \log p}{r^\beta}$ captures the segments with large popularities and large numbers of replicas (we choose $\alpha = \beta = 1$ in our simulations). These two kinds of segment replicas should be replaced by the replicas of segments with moderate popularities but a small number of copies. So in our system, we choose those segments with the smallest utility value as the victims to be replaced when a peer's cache is full. Thus, the data distribution is optimized naturally along with the progress of media accessing, and the efficiency of cache utilization is maximized.

3.4. Streaming Task Dispatch

In PROP, a streaming session is divided into a number of streaming tasks in a moderate granularity determined by the segment size (we use 100 KB in our simulation). These tasks can be served by different peers and it is the index server's responsibility to dispatch streaming tasks to different streaming servers. Instead of having multiple peers to collaboratively serve media steaming for a client like [9], only one streaming server is needed at a time in PROP. The failure of the streaming server has little impact on the client, and the media player only needs to buffer one segment for a smooth playback. Further more, the streaming tasks can be dispatched fairly and efficiently based on the information in segment indices and the quality of the streaming servers. Currently we only use the available bandwidth of the serving peer as the criterion to dispatch streaming tasks for load balance, and always dispatch streaming tasks to client peers first to decrease the proxy burden. We leave the dispatch optimization and data prefetching as a future work.

3.5. Fault Tolerance

When a peer fails, both the media data it caches and the segment indices it maintains are lost. In PROP, each peer periodically checks the validation of the replica location links in the segment indices it maintains and simply removes dead links. The loss of segment indices can be recovered by using the recovery mechanism of distributed hash table, e.g., CAN supports multiple realities to improve routing fault tolerance [10].

When the proxy fails or is overloaded so that it can not fetch data for clients, the requesting peer connects to the media server directly, fetches data and caches them locally until the proxy is recovered. Since the indices of media segments are distributed in the P2P system, the content locating mechanism still works and the system performance degrades gracefully. Compared with the solution of maintaining a central index in the proxy like the browser-aware proxy system [14], our system not only removes the single point of failure, but also significantly reduces the burdens of index maintenance and segment locating on the proxy.

4. Performance Evaluation

By using trace-driven simulation, we have comparatively evaluated the performance of our proposed system with different proxy cache sizes and different peer cache sizes and showed the different roles of the proxy and peers. When the total cache size of peers is zero, the system is equivalent to a *proxy caching system*. When the cache size of the proxy is zero, the system is equivalent to a *client-based P2P system* without proxy caching. In the following evaluations, if not specified, the default replacement policy on the proxy is popularity-based, and the default replacement policy on peers is utility-based for PROP system, or LRU for client-based P2P system, respectively.

We use the following metrics in our evaluations. The major metric, *streaming jitter byte ratio*, is defined as the data that is not served to the client in time by the proxy and peers, thus causing the potential playback jitter on the client side, normalized by the total bytes the clients demand. The second metric is the *delayed start request ratio*, which is defined as the number of requests suffering startup delays, normalized by the total number of requests. These two metrics reflect the QoS of the media streaming to the client. The third metric we use is the *byte hit ratio*, which is defined as the total bytes of media data served by the proxy and peers, normalized by the total bytes of media data all peers consume. Byte hit ratio represents the storage utilization and outgoing network traffic reduction of the system.

4.1. Workload Summary

Trace name	# of req.	# of obj.	# of peers	Size (GB)	λ	α	Range (min.)	Duration (day)
REAL	11559	400	1663	24	-	-	6-131	10
WEB	15188	400	376	51	4	0.47	2-120	1
PART	15188	400	376	51	4	0.47	2-120	1

Table 1. Workload Summary

Table 1 outlines the properties of the workloads. REAL denotes the workload traces extracted from the server logs of HP Corporate Media Solutions, covering the period from

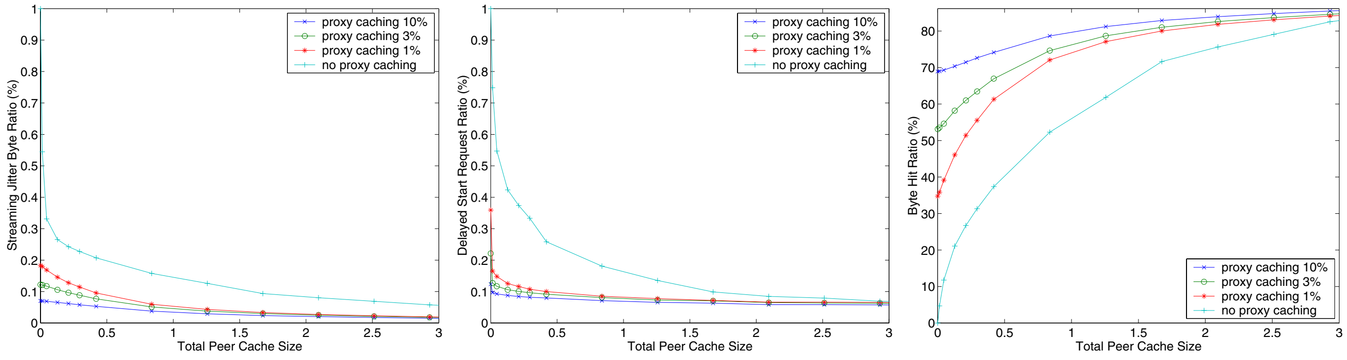

Figure 2. Performance evaluation on REAL workload. Left: Streaming jitter byte ratio; Middle: Delayed start request ratio; Right: Byte hit ratio.

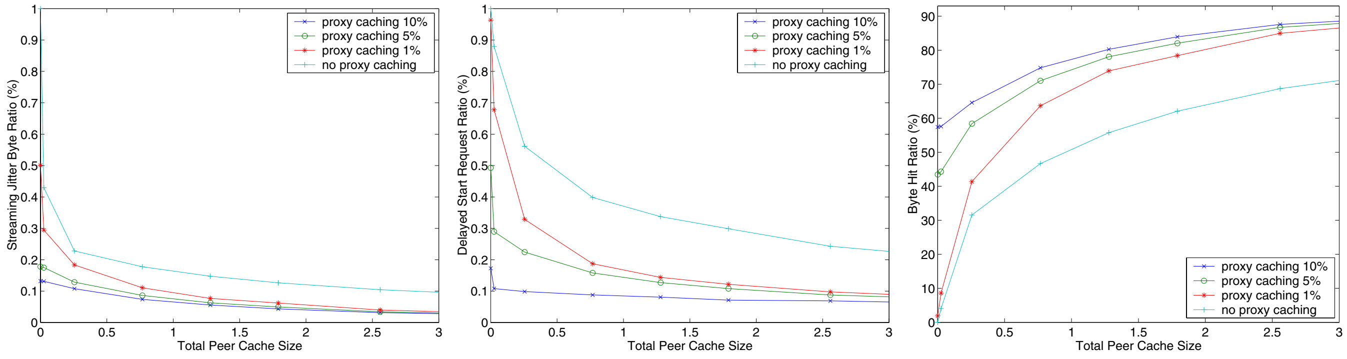

Figure 3. Performance evaluation on WEB workload. Left: Streaming jitter byte ratio; Middle: Delayed start request ratio; Right: Byte hit ratio.

May 1 through May 10, 2001. WEB and PART are two synthetic workloads for media viewing in the Web environment, where we assume the popularities of media objects follow Zipf-like distribution ($p_i = f_i / \sum_{i=1}^{N} f_i, f_i = 1/i^{\alpha}$) and the request arrivals follow Poisson distribution ($p(x, \lambda) = e^{-\lambda} * \lambda^x / x!, x = 0, 1, 2...$). WEB denotes complete viewing scenario while PART denotes partial viewing scenario in the Web environment. In PART, 80% of the requests only view 20% of the objects and then terminate.

For REAL workload, we set the bandwidth between the proxy and media server as the clients' available bandwidths in the trace, ranging from 0 bps to 6.248 Mbps. The media encoding bandwidth in REAL workload ranges from 6.9 Kbps to 2.055 Mbps. For synthetic workloads, we choose the bandwidth between the proxy and media server randomly from 0.5 to 2 times the encoding bandwidth of corresponding media objects. We assume the bandwidth of the intranet is broad enough for media streaming. We also assume each peer is online randomly, and can serve at most 5 clients at the same time. Each peer's cache size is proportional to the size of media content it accessed. For client-based P2P system model without proxy caching, each peer fetches media data individually while it caches its accessed data and publishes them in the system in the same way as our proposed system.

4.2. Performance Results

4.2.1. Overall performance Figure 2 shows the performance results of the three different system models (PROP, proxy caching, and client-based P2P system) on the REAL workload. We selected the proxy cache size as 10%, 3%, and 1% of all accessed media data in the workload, and ranged the total peer cache size from 0 to about 3 times the total accessed media data. In each figure, "no proxy caching" denotes the performance results of the client-based P2P system. The vertical axis denotes the performance results of the proxy caching system since the total peer cache size is zero. The results in Figure 2 show that our proposed system can significantly improve the QoS of media streaming and the byte hit ratio with the increase of peer cache size. For example, compared with the proxy caching system, our system can reduce up to 87% streaming jitter bytes as shown in Figure 2 (left), reduce up to 82% delayed start requests as shown in Figure 2 (middle), and increase the byte hit ratio up to 2.4 times as shown in Figure 2 (right). This is because more me-

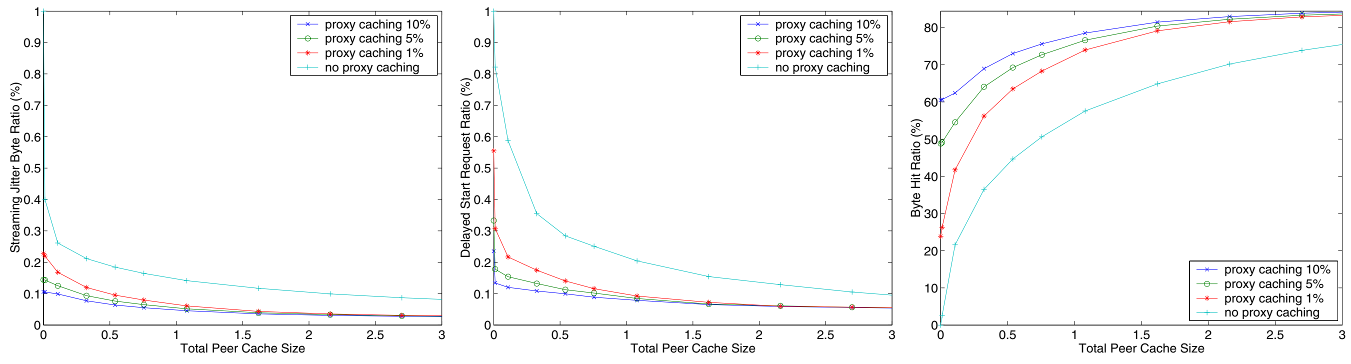

Figure 4. Performance evaluation on PART workload. Left: Streaming jitter byte ratio; Middle: Delayed start request ratio; Right: Byte hit ratio.

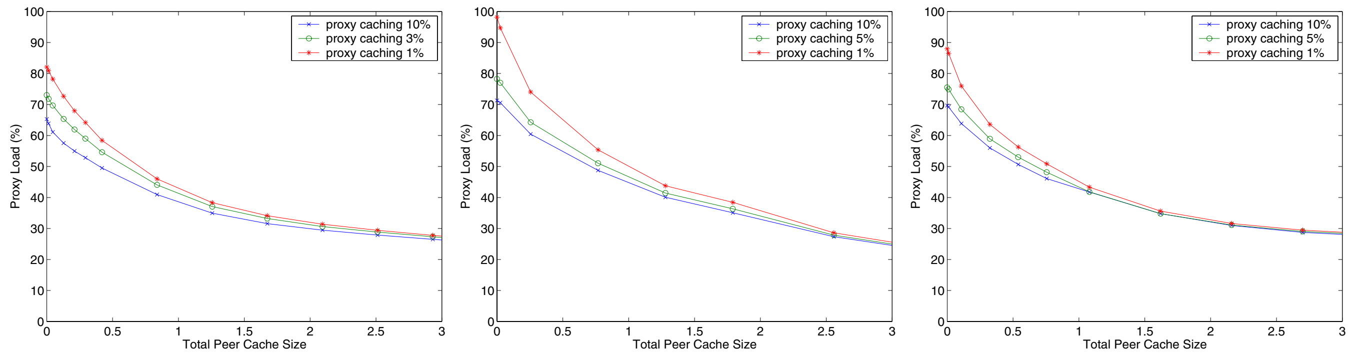

Figure 5. Proxy load changes. Left: with REAL workload; Middle: with WEB workload; Right: with PART workload.

dia data can be cached in the system due to the more storage contributed by peers, and thus improves both the byte hit ratio and QoS of media streaming correspondingly.

Figure 2 also shows that the proxy plays an important role in P2P media streaming system. The QoS and the byte hit ratio are improved significantly with the increase of the proxy cache size. Compared with client-based P2P system, our proposed system with a proxy capable of caching 1% of all media data can reduce up to 93% of streaming jitter bytes, up to 88% of delayed start requests, and the byte hit ratio can be as high as 85%, as shown in Figure 2 (left), 2 (middle), and 2 (right) respectively. In our system, the streaming performance can be effectively improved as we increase the proxy size. The reason is that a peer is not a dedicated server. It comes and leaves randomly and can only serve a small number of clients simultaneously. Thus, it is possible that a client can not be served immediately by peers that cache the requested data. On the other hand, the proxy provides a permanent cache and dedicated service to all clients; it takes over the streaming service when the peers are not capable of providing the streaming service, ensuring the quality of media streaming and reducing the outgoing bandwidth cost.

Figure 3 and Figure 4 show the performance results with the WEB workload and the PART workload respectively. As shown in these figures, the performances with these two workloads have similar trends to that with the REAL workload. The performance with workload WEB is slightly better than that with workload PART, which seems strange since there are more access localities in workload PART. This is because in WEB workload, the total amount of bytes consumed by all clients is about 2.8 times greater than that in PART workload while the total media objects are nearly the same as that in PART workload.

4.2.2. Proxy Load Changes

To evaluate the collaboration and coordination between the proxy and the P2P overlay, we have measured and observed the proxy load changes of our proposed system with different proxy cache sizes and peer cache sizes. The proxy load includes the total amount of bytes it serves peers and the total amount of bytes it fetches from the media server. Figure 5 (left), 5 (middle), and 5 (right) show the proxy load for the three workloads REAL, WEB, and PART respectively. In the figures, the proxy load is normalized by two times the total bytes all peers consume, the maximal possible load on the proxy. Increasing the cache size of peers can significantly reduce the proxy load due to the streaming service provided by peers in the system. The

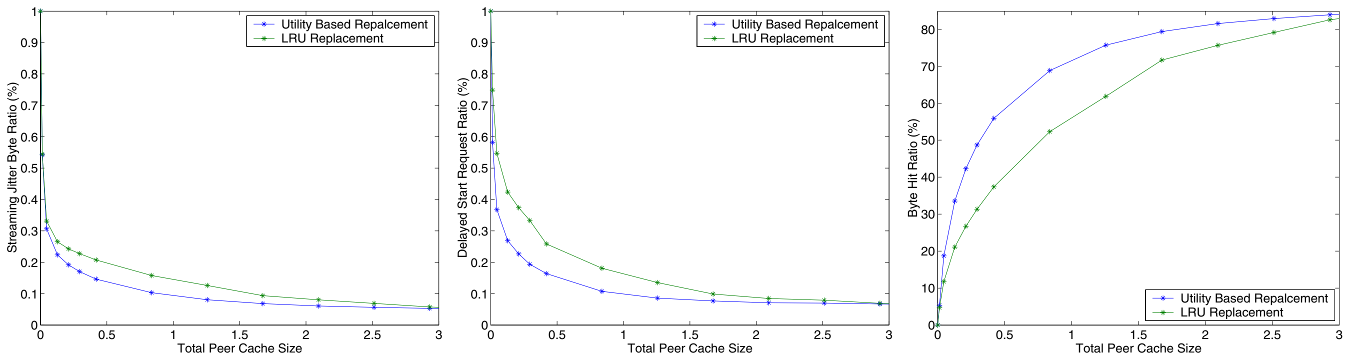

Figure 6. Replacement policy comparisons on REAL workload. Left: Streaming jitter byte ratio; Middle: Delayed start request ratio; Right: Byte hit ratio.

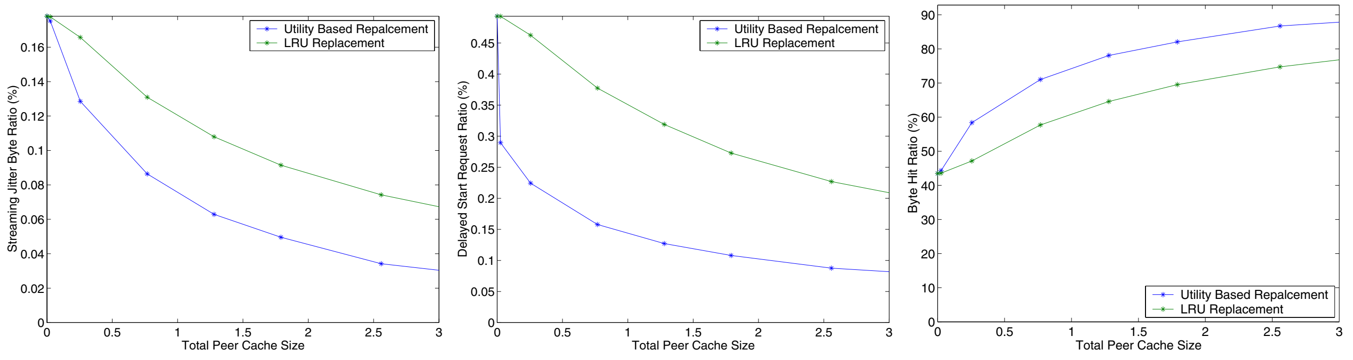

Figure 7. Replacement policy comparisons on WEB workload. Left: Streaming jitter byte ratio; Middle: Delayed start request ratio; Right: Byte hit ratio.

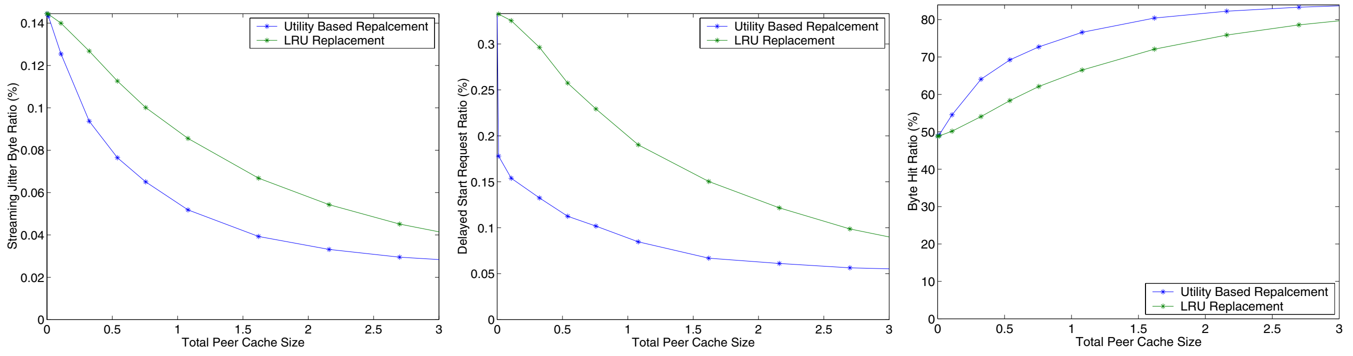

Figure 8. Replacement policy comparisons on PART workload. Left: Streaming jitter byte ratio; Middle: Delayed start request ratio; Right: Byte hit ratio.

proxy load reduction is up to 72% for REAL workload compared with the proxy caching system, as shown in Figure 5 (left). We also observe that the proxy load decreases with the increase of the proxy cache size. This is because when the proxy's cache size increases, more media data can be cached, which reduces the outgoing fetching load on the proxy.

4.2.3. Replacement Policy To evaluate the efficiency of resource management of our proposed system, we compared the performance of our global replacement policies with that of LRU replacement policy. Due to the page limit, we only present the comparison between the utility-based policy and LRU policy for peers, and only show the results without proxy caching for REAL workload and the results with proxy caching 5% of the total accessed media data for PART and WEB workloads. For other proxy cache sizes, similar performance results are achieved.

Figures 6, 7, and 8 compare the performance of the system using our utility-based peer replacement policy with that of the system using the LRU peer replacement policy, for the three workloads REAL, WEB, and PART, respectively. In both systems, the proxy uses the popularity-based replacement policy if it exists. We can see the streaming jitter bytes and delayed start requests of the utility-based replacement policy decrease much faster than that of the LRU replacement policy, while the byte hit ratio of the utility-based policy increases much faster than that of the LRU policy, indicating our utility-based replacement policy is significantly more effective than LRU policy. For REAL workload, compared with the system using LRU, the system using utility-based policy can reduce up to 36% streaming jitter bytes and up to 42% delayed start requests, and the improvement of byte hit ratio is as high as 59%, as shown in Figure 6 (left), 6 (middle) and 6 (right) respectively. For workload PART, the streaming jitter reduction and the delayed start request reduction is up to 41% and 56%, respectively, and the improvement of byte hit ratio is up to 19%. The byte hit ratio improvement in REAL workload is higher than that in PART and WEB workloads because there is no proxy caching in this case, showing the important role of proxy in PROP system. The utility-based replacement policy can greatly reduce delayed start requests with only a small cache size in each peer because the initial segments of media objects are generally more popular than the latter segments and have greater possibility to be cached. On the contrary, the initial segments are more likely to be replaced by LRU, with the progress of media viewing. At the same time, utility-based peer replacement policy takes into account both the popularities and the numbers of replicas of cached segments to maximize the storage utilization, therefore reduces the outgoing bandwidth consumptions and improves the QoS of media streaming.

5. Conclusion

Existing Internet streaming media delivering techniques are either based on a client-server model, such as proxy caching and server replications by CDNs, or based on a client-based P2P structure. The disadvantage of the client-server model is its limited scalability and high cost, while the disadvantage of a client-based P2P system is its unreliable quality of streaming media delivery due to the dynamic nature of peers. Our proposed system addresses these two limitations. Although there is a proxy in the system, our system structure is not centralized, because the proxy is a member of the P2P system managed by the distributed hash table. However, the proxy also plays an important and unique role to ensure the quality of media delivery due to its dedicated and stable nature. The collaboration and coordination between the proxy and client-based P2P system make the entire streaming media system both performance-effective and cost-effective. The proxy can also be used for other purposes to enhance the security of the system, such as to add the functions of anonymous communications and firewalls.

Acknowledgment: We thank Theresa Long and Robert Richard Painter for reading this paper and their constructive comments.

References

[1] Akamai Technologies Inc, "Delivering a better internet," http://www.akamai.com/.

[2] K. Wu, P. S. Yu, and J. Wolf, "Segment-based proxy caching of multimedia streams," in *Proceedings of the 10th International Conference on World Wide Web*, Hong Kong, China, May 2001, pp. 36–44.

[3] S. Chen, B. Shen, S. Wee, and X. Zhang, "Adaptive and lazy segmentation based proxy caching for streaming media delivery," in *Proceedings of the 13th ACM NOSSDAV*, Monterey, CA, June 2003, pp. 22–31.

[4] D. Tran, K. Hua, and T. Do, "Zigzag: An efficient peer-to-peer scheme for media streaming," in *Proceedings of IEEE INFOCOM 2003*, San Francisco, CA, April 2003.

[5] V. Padmanabhan, H. Wang, P. Chou, and K. Sripanidkulchai, "Distributing streaming media content using cooperative networking," in *Proceedings of the 12th ACM NOSSDAV*, Miami, Florida, May 2002, pp. 177–186.

[6] R. Rejaie and A. Ortega, "Pals: peer-to-peer adaptive layered streaming," in *Proceedings of the 13th ACM NOSSDAV*, Monterey, CA, June 2003, pp. 153–161.

[7] Y. Cui and K. Nahrstedt, "Layered peer-to-peer streaming," in *Proceedings of the 13th ACM NOSSDAV*, Monterey, CA, June 2003, pp. 162–171.

[8] X. Jiang, Y. Dong, D. Xu, and B. Bhargava, "Gnustream: A p2p media streaming system prototype," in *Proceedings of the 4th International Conference on Multimedia and Expo*, Baltimore, Maryland, July 2003.

[9] M. Hefeeda, A. Habib, B. Botev, D. Xu, and B. Bhargava, "Promise: A peer-to-peer media streaming system," in *Proceedings of the 11th Annual ACM International Conference on Multimedia*, Berkeley, CA, November 2003.

[10] S. Ratnasamy, P. Francis, M. Handley, and R. Karp, "A scalable content-addressable network," in *Proceedings of ACM SIGCOMM 2001*, San Diego, CA, August 2001, pp. 161–172.

[11] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proceedings of ACM SIGCOMM 2001*, San Deigo, CA, August 2001, pp. 149–160.

[12] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in *Proceedings of IFIP/ACM Middleware 2001*, Heidelberg, Germany, November 2001, pp. 329–350.

[13] B. Zhao, J. Kubiatowicz, and A. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing," in *Report No. UCB/CSD-01-1141*, April 2001.

[14] L. Xiao, X. Zhang, and Z. Xu, "On reliable and scalable peer-to-peer web document sharing," in *Proceedings of 2002 International Parallel and Distributed Processing Symposium*, Fort Lauderdale, Florida, April 2002.