Proxy Caching for Media Streaming over the Internet

Jiangchuan Liu
School of Computing Science
Simon Fraser University
Burnaby, BC, Canada
csljc@ieee.org

Jianliang Xu
Department of Computer Science
Hong Kong Baptist University
Hong Kong
xujl@comp.hkbu.edu.hk

ABSTRACT

Streaming media has contributed to a significant amount of today's Internet traffic. Like conventional web objects (*e.g.*, HTML pages and images), media objects can benefit from proxy caching; yet their unique features such as huge size and high bandwidth demand imply that conventional proxy caching strategies have to be substantially revised. This article discusses the critical issues and challenges of cache management for proxy-assisted media streaming. We survey, classify, and compare the state-of-the-art solutions. We also investigate advanced issues of combining multicast with caching, cooperating among proxies, and leveraging proxy caching in overlay networks.

Keywords: *Streaming Media, Proxy, Caching, Overlay Networks*

1. INTRODUCTION

With widespread penetration of the broadband Internet, multimedia service is getting increasingly popular among users and has contributed to a significant amount of today's Internet traffic. Media objects can be accessed similar to conventional text and images using a download-and-play mode; but most users prefer to quickly initiate and then continuously play back a media object while it is being downloaded, *i.e.*, to use a *real-time streaming* mode. We have witnessed the initial and incremental deployment of streaming applications like RealNetworks RealPlayer and Microsoft Windows Media Player in recent years. The performance of such applications however is still far from satisfactory, especially during the peak hours.

To reduce client-perceived access latencies as well as server/network loads, an effective means is to cache frequently used data at proxies close to clients. It also enhances the availability of objects and mitigates packet losses, as a local transmission is generally more reliable than a remote transmission. Proxy caching thus has become one of the vital components in virtually all web systems. Streaming media, particularly those pre-stored, could also benefit significant performance improvement from proxy caching, given their static nature in content and highly localized access interests. However, existing proxies are generally optimized for delivering conventional web objects (*e.g.*, HTML pages or GIF images), which may not meet the requirements of steaming applications. In the following, we list some important and unique features of streaming media and discuss their implications to proxy cache design.

Huge size: A conventional web object is typically on the order of 1K to 100K bytes. Hence, a binary decision works well for proxy caching: either caching an object in its entirety or not caching. In contrast, a media object has a high data rate and a long playback duration, which combined yield a huge data volume. For illustration, a one-hour standard MPEG-1 video has a volume of about 675 MB; caching it entirely at a web proxy is clearly impractical, as several such large streams would exhaust the capacity of the cache. One solution is to cache only portions of an object. In this case, a client's playback needs a joint delivery involving both the proxy and the origin server. To cache which portions of which objects thus has to be carefully managed, such that the benefit of caching outweighs the synchronization overhead of the joint delivery.

Intensive bandwidth use: Streaming nature of delivery requires a significant amount of disk and network I/O bandwidth, sustaining over a long period. Hence, minimizing bandwidth consumption becomes a primary consideration for proxy cache management, even taking precedence over reducing access latencies in many cases. Moreover, the bandwidth bottleneck

limits the number of clients that a proxy can simultaneously support; employing multicast delivery and cooperation among proxies thus become particularly attractive for media streaming applications.

High interactivity: The long playback duration of a streaming object also enables various client-server interactions. As an example, recent studies found that nearly 90% media playbacks are terminated prematurely by clients [1]. In addition, during a playback, a client often expects VCR-like operations, such as fast-forward and rewind. This implies the access rates might be different for different portions of a stream, which potentially complicates the cache management.

Given these unique features of media objects, novel caching algorithms have been developed in the literature. The objective of this article is to review the state-of-the-art caching techniques dedicated to streaming media caching. We begin with discussions on a generic proxy caching architecture and some protocol considerations. The caching strategies for streaming media are classified, examined, and compared in Section 3. Section 4 investigates some advanced issues. Finally, Section 5 concludes the article.

2. ARCHITECTURE AND PROTOCOLS FOR STREAMING CACHING

Streaming applications generally support diverse client-server interactions and have stringent demands on packet delay and jitter to ensure discontinuity-free playback. To meet these requirements, the Internet Engineering Task Force (IETF) has developed the RTP/RTCP/RTSP protocol suite. A generic system diagram of proxy-assisted media streaming using this suite is depicted in Fig. 1.

In this system, the basic functionalities for data transferring are provided by the Real-Time Transport Protocol (RTP), including payload identification, sequence numbering for loss detection, and time stamping for playback control. Running on top of UDP, RTP itself does not guarantee Quality-of-Service (QoS), but relies on its companion, the Real-Time Control Protocol (RTCP), to monitor the network status and provide feedback for application-layer adaptation. The Real-Time Streaming Protocol (RTSP) coordinates the delivery of media objects and enables a rich set of controls for interactive playback. For the proxy-assisted streaming, the proxy has to relay these control messages between the client and the server. The problem is particularly involved if only part of a media object is cached at a proxy. In this case, the proxy must reply to the client PLAY request and initiate transmission of RTP and RTCP messages to the client for the cached portion, while request the uncached portion(s) from the server. Such fetching can be

achieved through an RTSP Range request specifying the playback points. The Range request also enables clients to retrieve different segments of a media object from multiple servers or proxies, if needed.

Beside this classical client/server paradigm, peer-to-peer streaming and other overlay streaming paradigms have also attracted much attention recently, which will be discussed in Section 4.C.

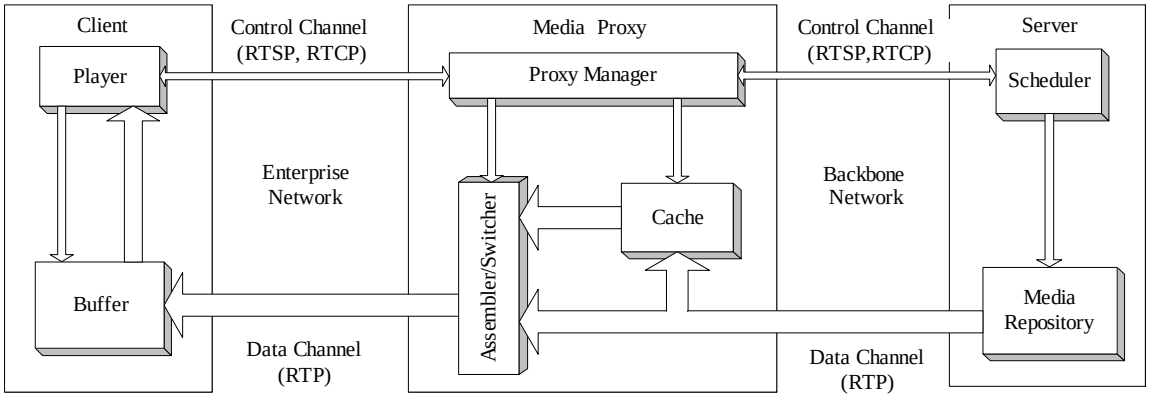


Figure 1. A generic system diagram of proxy-assisted media streaming using RTP/RTCP/RTSP.

3. CACHING STRATEGIES FOR STREAMING MEDIA

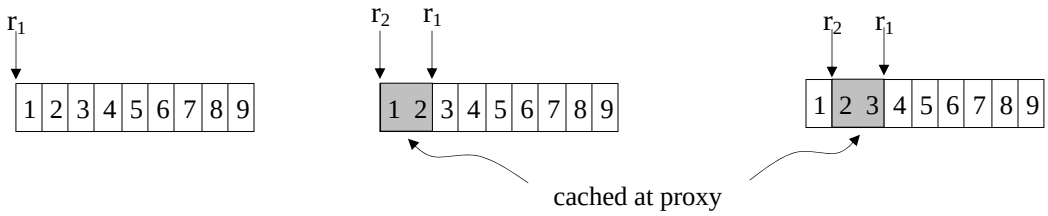
Due to the aforementioned features of streaming media objects, media caching has many distinct focuses from conventional web caching. On one hand, since the content of a media object is rarely updated, management issues like cache consistency and coherence are less critical in media caching. On the other hand, given high resource requirements of media objects, effective management of proxy cache resources (*i.e.*, space, disk I/O, and network I/O) becomes more challenging. In this section, we survey the state-of-the-art media caching strategies for both homogenous clients and heterogeneous clients, with an emphasis on how the strategies minimize resource demands.

A. Stream Caching for Homogeneous Clients

Most existing caching algorithms focus on homogeneous clients, which have identical or similar configurations and capabilities behind a proxy. As such, a single version of an object would match the bandwidth and format demands of all requests to the object. Nevertheless, what to cache (which portions of which objects) and how to manage cache (*e.g.*, cache placement and replacement) at the proxy remain challenges. According to the selection of the portions to cache,

we classify existing algorithms into four categories: *sliding-interval caching*, *prefix caching*, *segment caching*, and *rate-split caching*.

Sliding-Interval Caching [2]: This algorithm caches a sliding interval of a media object to exploit sequential access of streaming media. For illustration, given two consecutive requests for the same object, the first request may access the object from the server and incrementally store it into the proxy cache; the second request can then access the cached portion and release it after the access. If the two requests arrive close in time, only a small portion of the media object needs to be cached at any time instance, and yet the second request can be completely satisfied from the proxy (see Fig. 2). In general, if multiple requests for an object arrive in a short period, a set of adjacent intervals can be grouped to form a *run*, of which the cached portion will be released only after the last request has been satisfied.



(a) Time 0: request 1 arrives

(b) Time 1-2: frames 1 and 2 accessed by request 1 and cached; request 2 arrives.

(c) Time 2-3: frame 3 accessed by request 1 and cached; frame 1 read by request 2 and released.

Figure 2. An illustration of sliding-interval caching. The object consists of 9 frames, each requiring one unit time to deliver from the proxy to a client. Requests 1 and 2 arrive at times 0 and 2, respectively. To serve request 2, only two frames need to be cached at any time instance.

Sliding-interval caching can significantly reduce network bandwidth consumption and start-up delay for subsequent accesses. However, as the cached portion is dynamically updated with playback, the sliding-interval caching involves high disk bandwidth demands; in the worse case, it would double the disk I/O due to the concurrent read/write operations. In addition, its effectiveness diminishes with the increase of the access intervals. If the access interval of the same object is longer than the duration of the playback, the algorithm is degenerated to the unaffordable full-object caching. To address these issues, it is preferable to retain the cached content over a relatively long time period. Most of the caching algorithms to be discussed in the rest of this section fall into this category.

Prefix Caching [3]: This algorithm caches the initial portion of a media object, called *prefix*, at a proxy. Upon receiving a client request, the proxy immediately delivers the prefix to the client and, meanwhile, fetches the remaining portion, the *suffix*, from the server and relays to the client (see Fig. 3). As the proxy is generally closer to the clients than the origin server, the start-up delay for a playback can be remarkably reduced.

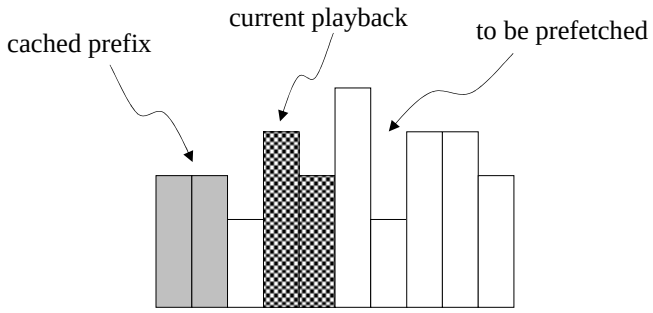


Figure 3. A snapshot of prefix caching with workahead smoothing.

To ensure discontinuity-free playback with a start-up delay of *s*, the proxy has to store a prefix of length $\max\{d_{\max} - s, 0\}$, where $d_{\max}$ is the maximum delay from the server to the proxy. If cache space is abundant, the proxy can also devote some space to assist in performing *workahead smoothing* for variable-bit-rate (VBR) media [3]. With this smoothing cache, the proxy can prefetch large frames in advance of each burst to absorb delay jitter and bandwidth fluctuations of the server-to-proxy path. The delay of prefetching can be hided by the prefix caching. Similar to sliding-interval caching, the content of the smoothing cache is dynamically updated with playback. However, the purposes are different: the former is to improve cache hit for subsequent requests, while the latter is to facilitate workahead smoothing.

Segment Caching [1, 4, 5, 6]: Segment caching generalizes the prefix caching paradigm by partitioning a media object into a series of segments, differentiating their respective utilities, and making caching decision accordingly (see Fig. 4). Various segment caching algorithms have been proposed in the literature by employing different segmentations and utility calculations. Wu *et al.* [4] suggested to group the frames of a media object into variable-sized segments, with the length increasing exponentially with the distance from the start of the media, *i.e.*, the size of segment i is 2^{i-1}, which consists of frames 2^{i-1}, $2^{i-1}+1$, …, $2^{i}-1$. The motivation is that a proxy can quickly adapt to the changing access patterns of cached objects by discarding big chunks as needed. The utility of a segment is calculated as the ratio of the segment reference frequency over its distance from the beginning segment, which favors to cache the initial segments as well as

those with higher access frequencies. Chen *et al*. [1], however, argued that neither the use of a predefined segment length nor the favorable caching of the initial segments is the best strategy for reducing network traffic. They suggested postponing segmentation as late as possible (called *lazy segmentation*), thus allowing the proxy to collect a sufficient amount of access statistics to improve the effectiveness.

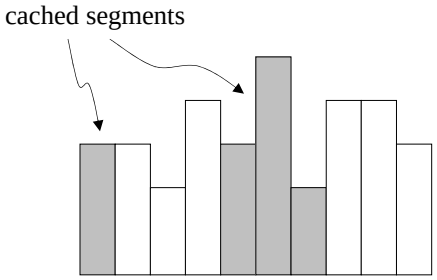


Figure 4. An illustration of segment caching.

A salient feature of segment-based caching is its support to VCR-like operations, such as random access, fast-forward, and rewind. As an example, Fahmi *et al.* [6] proposed to cache some key segments of a media object, called *hotspots*, which are identified by content providers. When a client requests the object, the proxy first delivers the hotspots to provide an overview of the stream; the client can then decide whether to play the entire stream or quickly jump to some specific portion introduced by a hotspot. Furthermore, in fast-forwarding and rewinding operations, only the corresponding hotspots are delivered and displayed, while other portions are skipped. As such, the load of the server and backbone network can be greatly reduced, but the client will not miss any important segments in the media object.

Rate-Split Caching [7]: While all the aforementioned caching algorithms partition a media object horizontally along the time axis, the rate-split caching partitions a media vertically along the rate axis: the upper part will be cached at the proxy, whereas the lower part will remain stored at the origin server (see Fig. 5). This type of partitioning is particularly attractive for VBR streaming, as only the lower part of a nearly constant rate has to be delivered through the backbone network. For a QoS network with resource reservation, the bandwidth reserved should be equal to the peak rate of a stream; caching the upper part at the proxy clearly reduces the rate variability and improves the backbone bandwidth utilization. A critical issue here is how to select the cut-off rate or, equivalently, the size of the upper part for caching. Zhang *et al*. [7] studied the impact of the cut-off rate for a single stream through empirical evaluation, and found that a significant bandwidth reduction can be achieved with a reasonably small cache space. They also

formulated the multiple-stream case as a knapsack problem with two constraints: disk bandwidth and cache space, and developed several heuristics, *e.g.*, caching popular objects only, or caching those with high bandwidth reduction.

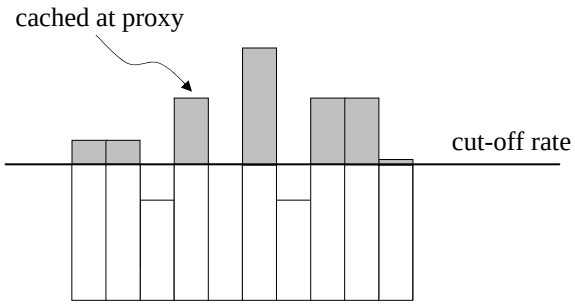


Figure 5. An illustration of rate-split caching.

Summary and Comparison: Table 1 summarizes the caching algorithms reviewed for homogeneous clients. While these features and metrics provide a general guideline for algorithm selection, the choice for a specific streaming system also largely depends on a number of practical issues, in particular, the complexity of the implementation. In fact, only a few simple algorithms have been employed in commercial systems, though recently-built prototypes have practically demonstrated the viability and superiority of the intelligent algorithms, such as *lazy segmentation* [1].

TABLE I. COMPARISONS OF THE CACHING ALGORITHMS FOR HOMOGENEOUS CLIENTS

		Sliding-Interval Caching [2]	Prefix Caching [3]	Segment Caching [1,4,5,6]	Rate-Split Caching [7]
Cached Portion		Sliding intervals	Prefix	Segments	Portion of higher rate
VCR-like support		No	No	Yes	No
Resource Demand	Disk I/O	High	Moderate	Moderate	Moderate
	Disk Space	Low	Moderate	High	High
	Sync Overhead	Low	Moderate	High	High
Performance Improvement	Bandwidth Reduction	High*	Moderate	Moderate	Moderate
	Start-up Latency Reduction	High*	High	High**	Moderate

* There is no reduction for the first request in a run.
** Assume the initial segment is cached.

In addition, we emphasize that these algorithms are not necessarily exclusive with each other, and a combination of them may yield a better performance. For example, segment caching combined with prefix caching of each segment can reduce start-up latency for VCR-like random playback from any key-segment. Combination with conventional data caching algorithms has also been examined.

B. Stream Caching for Heterogeneous Clients

Owing to diverse network models and device configurations, clients behind the same proxy often have quite different requirements on the same media object, in terms of streaming rates or encoding formats. To accommodate such heterogeneity, a straightforward solution is to produce replicated streams of different rates or formats, each targeting on a subset of clients. Though being widely used in commercial streaming system, the storage and bandwidth demands of this approach can be prohibitively high [8]. An alternative is to transcode a media from one form to another of a lower rate or a different encoding format in an on-demand fashion [9]. The intensive computation overhead of transcoding however prevents a proxy from supporting a large, diverse client population.

Yet a more efficient approach to this problem is the use of layered encoding and transmission. A layered coder compresses a raw media object into several layers: the most significant layer, called the *base layer*, contains the data representing the most important features of the object, while additional layers, called *enhancement layers*, contain the data that can progressively refine the quality. A client thus can subscribe to a subset of cumulative layers to reconstruct a stream commensurate with its capability. For layered caching, Kangasharju *et al.* [10] assumed that the cached portions are semi-static and only completed layers are cached. To maximize the total revenue, they developed effective heuristics based on an analytical stochastic knapsack model to determine the cache content. In their model, the client population and the distribution of their capacities are known *a priori*. For layered streaming under dynamic conditions, Rejaie *et al.* [11] studied segment-based cache replacement and prefecting policies to achieve efficient utilization of cache space and available bandwidth (see Fig. 6a). The main objective is to deal with the congestion problem for individual clients. To this end, the proxy keeps track of popularities of each object on a per layer basis. When the quality of the cached layers is lower than the maximum deliverable quality to an interested client, the proxy sends requests to the server for missing segments within a sliding prefetching window. On cache replacement, a victim layer is

identified based on popularities, and its cached segments are flushed from the tail until sufficient space is obtained.

A critical drawback of the existing layered streaming systems is that the number of layers is pretty small, typically 2 or 3 only; hence, their adaptation granularity remains coarse. Fortunately, recent development in the coding area has demonstrated the possibility of fine-grained post-encoding rate control. An example is the MPEG-4 Fine-Grained Scalable (FGS) coder with bitplane coding, which generates embedded streams containing several bitplanes and each can be partitioned at any specific rate. As such, for narrowband clients, the proxy can reduce the streaming rate using a bitplane filter; for wideband clients, the proxy can fetch some uncached portion (*i.e.*, higher-order bitplanes) from the server and assemble it with the cached portion to generate a high-rate stream. As illustrated in Fig. 6b, the available bandwidth of a client can be almost fully utilized, and, more importantly, both the filtering and the assembling operations in FGS can be done with fast response. Hence, we envision the FGS-based streaming and caching as a very promising solution to media steaming over the Internet comprising highly heterogeneous end-systems. Several caching algorithms have been proposed to minimize the bandwidth consumption and/or improve the client utility [8].

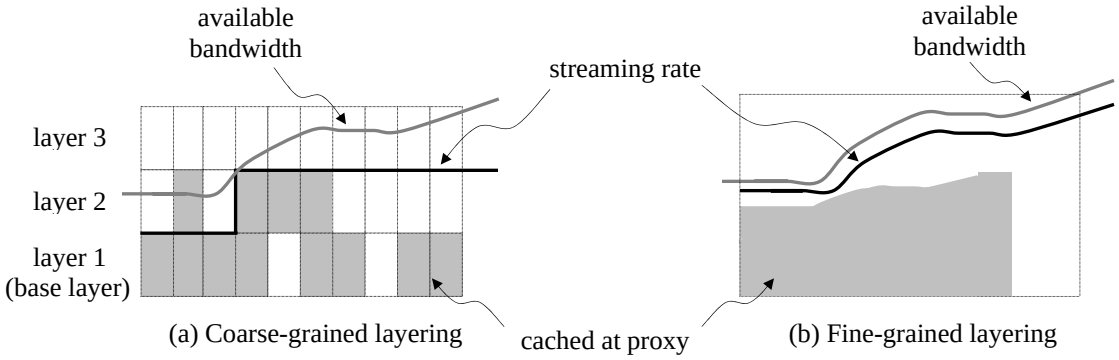


Figure 6. Caching for layered streaming. (a) Coarse-grained layering; (b) Fine-grained layering.

4. ADVANCED ISSUES

So far we consider a standalone proxy with only unicast delivery. While it can noticeably reduce the access latencies and backbone bandwidth demands, the scalability and robustness of this simple architecture are still restricted. We now discuss two effective enhancements: multicast and proxy cooperation; we also address the role of proxy in the recently popularized overlay communication paradigms.

A. Combining Proxy Caching with Multicasting

Like caching, multicasting also explores the temporal locality of client requests. Specifically, it allows a media server to accommodate concurrent client requests with shared channels through batching, patching, or periodic broadcast. However, multicast delivery suffers from two important deficiencies. First, to save more bandwidth, it is better to accommodate more requests in one multicast channel by using a large batching/patching window, which however leads to long start-up latencies. Second, while IP multicast is enabled in virtually all local-area networks, its deployment over the global Internet remains limited in scope and reach. Hence, it is unlikely that a multicast streaming protocol can be used for geographically dispersed servers and clients.

Interestingly, both deficiencies can be alleviated through the use of proxies. Specifically, a request can be instantaneously served by a cached prefix while waiting for the data from a multicast channel [12,13], and proxies can bridge unicast networks with multicast networks, *i.e.*, employing unicast for server to proxy delivery while batching and/or patching local accesses. Wang *et al*. [13] have derived the optimal length of the prefix to be cached for most typical multicast protocols, and showed that a careful coupling of caching and multicasting can produce significant cost savings over using the unicast service, even if IP multicast is supported only at local networks.

B. Cooperative Proxy Caching

In general, proxies grouped together can achieve better performance than independent standalone proxies. Specifically, the group of proxies can cooperate with each other to increase the aggregate cache space, balance loads, and improve system scalability [14,15]. A typical cooperative media caching architecture is MiddleMan [14], which operates a collection of proxies as a scalable cache cluster. Media objects are segmented into equal-sized segments and stored across multiple proxies, where they can be replaced at a granularity of a segment. There are also several local proxies responsible to answer client requests by locating and relaying the segments. Note that, in cooperative web caching, a critical issue is how to efficiently locate web pages with minimum communication costs among the proxies. This is, however, not a major concern for cooperative media caching, as the bandwidth consumption for streaming objects is of orders of magnitude higher than that for object indexing and discovering. Consequently, in MiddleMan, a centralized coordinator works well in keeping track of cache states. On the other hand, while segment-based caching across different proxies facilitates the distribution and balance of proxy loads, it incurs a significant amount overhead for switching among proxies to reconstruct a media

object. To reduce such effects as well as to achieve better load balance and fault tolerance, Chae *et al*. [15] suggested a Silo data layout, which partitions a media object into segments of increasing sizes, stores more copies for popular segments, but still guarantees at least one copy stored for each segment.

C. Streaming Caching in Overlay Networks

So far, we have focused on the client/server paradigm for media streaming, and proxies act as intermediaries between them. Generalizing the proxy functionalities into every end-host will shift the system to the recently popularized overlay communication paradigms, such as peer-to-peer communication or application-layer multicast. There have been many pioneering efforts on overlay streaming, which have demonstrated the superior scalability and deployability of these overlay systems; the enormous buffer capacities distributed in end-hosts also enable efficient client-side caching and sharing to improve content availability as well as to support asynchronous streaming [16, 17, 18, 19].

Nevertheless, we are aware that, in contrast to the reliable and dedicated servers or proxies, the loosely-coupled autonomous end-hosts can easily crash, leave without notice, or even refuse to share its own data. Given that a media playback lasts a long time and consumes huge resources, we believe dedicated proxies will still play an important role in building high-quality media streaming systems; in particular, strategically placed proxies may effectively assist the construction and maintenance of large-scale overlays. On the other hand, we may also leverage the overlay paradigm in proxy design. As an example, Guo *et al.* [20] suggested a proxy and its clients be structured into a peer-to-peer system to collaboratively serve local streaming requests. Their work focused on local area collaboration. We believe that it can be extended to a two-level streaming overlay: a cluster of proxies forms an overlay in wide-area networks while each proxy collaborates with local client caches to form a local overlay. These two overlays well complement each other. The proxy-level overlay provides a dedicated storage and reliable service, while the local overlay provides a scalable storage for caching and significantly reduce the load of the proxy.

5. CONCLUDING REMARKS

Proxy caching is an effective means to reduce access latencies as well as resource consumptions for networked applications. Due to the unique features of media objects like huge size and high bandwidth demand, a number of novel streaming caching solutions have been

reported in the literature. This article serves as a pioneer survey to this field, though it by no means covers all aspects. Plenty of research issues have yet to be addressed, *e.g.*, caching over the wireless mobile Internet, for large-scale dynamic overlays, and with advanced video coding scheme such as multiple description coding [18], as well as security and privacy for cached media objects, to name but a few. We envision that streaming media caching remains a fertile area, and both theoretical and practical solutions to the listed problems are urged with rising demands on ubiquitous multimedia services throughout the world.

REFERENCES

[1] S. Chen, B. Shen, S. Wee, and X. Zhang, "Designs of high quality streaming proxy systems," in *Proc. IEEE INFOCOM'04*, Hong Kong, Mar. 2004.

[2] R. Tewari, H. M. Vin, A. Dan, and D. Sitaram, "Resource-based caching for Web servers," in *Proc. SPIE/ACM Conf. on Multimedia Computing and Networking* (*MMCN'98*), San Jose, CA, Jan. 1998.

[3] S. Sen, J. Rexford, and D. Towsley, "Proxy prefix caching for multimedia streams," in *Proc. IEEE INFOCOM'99*, New York, NY, Mar. 1999.

[4] K. L. Wu, P. S. Yu, and J. L. Wolf, "Segment-based proxy caching of multimedia streams," in *Proc. World Wide Web Conference (WWW10)*, Hong Kong, May 2001.

[5] Z. Miao and A. Ortega, "Scalable proxy caching of video under storage constraints," *IEEE J. Select. Areas in Comm.*, vol. 20, no. 7, pp. 1315-1327, Sep. 2002.

[6] H. Fahmi, M. Latif, S. Sedigh-Ali, A. Ghafoor, P. Liu, and L. Hsu, "Proxy servers for scalable interactive video support," *IEEE Computer*, 43(9): 54-60, Sep. 2001.

[7] Z.-L. Zhang, Y. Wang, D. Du, and D. Su, "Video staging: A proxy-server-based approach to end-to-end video delivery over wide-area networks," *IEEE/ACM Trans. Networking*, 8(4): 429-442, 2000.

[8] J. Liu, X. Chu, and J. Xu, "Proxy Cache Management for Fine-Grained Scalable Video Streaming," *Proc. IEEE INFOCOM'04*, Hong Kong, Mar. 2004.

[9] X. Tang, F. Zhang, and S. T. Chanson, "Streaming media caching algorithms for transcoding proxies," in *Proc. 31st Int. Conf. on Parallel Processing* (*ICPP'*02), Aug. 2002.

[10] J. Kangasharju, F. Hartanto, M. Reisslein, and K. W. Ross, "Distributing layered encoded video through caches," *IEEE Trans. Computers*, 51(6), pp. 622-636, June 2002.

[11] R. Rejaie, H. Yu, M. Handley, and D. Estrin, "Multimedia proxy caching mechanism for quality adaptive streaming applications in the Internet," in *Proc. IEEE INFOCOM'00*, Tel Aviv, Israel, Mar. 2000.

[12] S. Ramesh, I. Rhee, and K. Guo, "Multicast with cache (Mcache): An adaptive zero-delay video-on-demand service," in *Proc. IEEE INFOCOM'01*, Anchorage, AK, Apr. 2001.

[13] B. Wang, S. Sen, M. Adler, and D. Towsley, "Optimal proxy cache allocation for efficient streaming media distribution," in *Proc. IEEE INFOCOM'02*, New York, NY, June 2002.

[14] S. Acharya and B. C. Smith, "Middleman: A video caching proxy server," in *Proc. 10th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'00)*, June 2000.

[15] Y. Chae, K. Guo, M. M. Buddhikot, S. Suri, and E. W. Zegura, "Silo, rainbow, and caching token: Schemes for scalable, fault tolerant stream caching," *IEEE J. Select. Areas in Comm.*, 20(7), pp. 1328-1344, Sep. 2002.

[16] D. Xu, M. Hefeeda, S. Hambrusch, and B. Bhargava, "On peer-to-peer media streaming," in *Proc. IEEE International Conference on Distributed Computing Systems (ICDCS'02)*, Wien, Austria, July 2002.

[17] S. Jin and A. Bestavros, "Cache-and-relay streaming media delivery for asynchronous clients," in *Proc. the 4th International Workshop on Networked Group Communication (NGC)*, Boston, MA, USA, Oct. 2002.

[18] V. N. Padmanabhan, H. J. Wang, P. A. Chou, and K. Sripanidkulchai, "Distributing streaming media content using cooperative networking," in *Proc. 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'02), Miami, FL, USA, May 2*002.

[19] Y. Cui, B. Li, and K. Nahrstedt, "oStream: Asynchronous Streaming Multicast in Application-Layer Overlay Networks," *IEEE J. Select. Areas in Comm.*, 22(1), pp. 91-106, Jan. 2004.

[20] L. Guo, S. Chen, S. Ren, X. Chen, and S. Jiang, "PROP: a scalable and reliable P2P assisted proxy streaming system," in *Proc.IEEE International Conference on Distributed Computing Systems* (*ICDCS'04)*, Tokyo, Japan, Mar. 2004.