SProxy: A Caching Infrastructure to Support Internet Streaming

Songqing Chen, *Member, IEEE*, Bo Shen, *Senior Member, IEEE*, Susie Wee, and Xiaodong Zhang, *Senior Member, IEEE*

***Abstract*—Many algorithmic efforts have been made to address technical issues in designing a streaming media caching proxy. Typical of those are segment-based caching approaches that efficiently cache large media objects in segments which reduces the startup latency while ensuring continuous streaming. However, few systems have been practically implemented and deployed. The implementation and deployment efforts are hindered by several factors: 1) streaming of media content in complicated data formats is difficult; 2) typical streaming protocols such as RTP often run on UDP; in practice, UDP traffic is likely to be blocked by firewalls at the client side due to security considerations; and 3) coordination between caching discrete object segments and streaming continuous media data is challenging. To address these problems, we have designed and implemented a segment-based streaming media proxy, called *SProxy*. This proxy system has the following merits. First, *SProxy* leverages existing Internet infrastructure to address the flash crowd. The content server is now free of the streaming duty while hosting streaming content through a regular Web server. Thus, UDP based streaming traffic from *SProxy* suffers less dropping and no blocking. Second, *SProxy* streams and caches media objects in small segments determined by the object popularity, causing very low startup latency, and significantly reducing network traffic. Finally, prefetching techniques are used to pro-actively preload uncached segments that are likely to be used soon, thus providing continuous streaming. *SProxy* has been extensively tested and we show that it provides high quality streaming delivery in both local area networks and wide area networks (e.g., between Japan and the U.S.).**

***Index Terms*—Content distribution, streaming, proxy caching, segmentation.**

I. Background and Motivation

THE DEMAND of robust, cost-effective, and high-quality Internet streaming services keeps increasing. It has been predicted that the amount of streaming media content will increase rapidly in the next a few years [1], [2] and its traffic will become a significant portion on the Internet. For example, compared the workload of a 2002 study [3] with the workload of a 1999 study [4] in a similar campus environment, the portion of network bytes ascribed to audio and video increased by 300% and 400% [5], respectively.

On the other hand, the delivery of diverse streaming media contents on IP networks in a cost effective manner, while maintaining high quality, is still very challenging. Thus, today most of Internet media objects are still accessed via downloading or pseudo streaming instead of streaming, which cause roughly 56% and 32% of wasted bandwidth according to study [6]. In a web service environment, a continuous streaming session (often with a duration of minutes or hours, compared to milliseconds or seconds for traditional Web pages) keeps consuming network bandwidth and disk bandwidth on the hosting server. Multiple concurrent streaming sessions can easily exhaust the available network bandwidth and overload the media content server [7]. Placing multimedia objects closer to clients is an effective solution that will relieve the network bottleneck and reduce the load on the media content server.

Research efforts have been made to extend existing proxy caching methods of static Web pages to the case of streaming media objects. Streaming media objects have some features that make caching promising: the objects are generally static and do not change for a long time. Moreover, they show some degree of locality of reference. However, proxy caching of multimedia objects also poses several challenges.

1) The size of media objects is usually several orders of magnitudes larger than traditional Web objects. For example, a 1-h movie encoded using MPEG4, at desktop resolution, may require more than 1 GB storage space. This limits the number of entire objects that can be stored on a caching server. It also results in large startup latencies if the object is not already cached.
2) Multimedia objects generally have very stringent demands in terms of continuous and timely delivery. This is especially challenging on the current Internet, which only provides best-effort services.
3) Prior research has observed that most of the media objects are only partially viewed [8], [9]. Using traditional, static web caching techniques to cache these large objects thus wastes storage and causes unnecessary network traffic.

To handle these problems, researchers have proposed various streaming caching approaches to support cost-effective and robust Internet streaming, by caching media objects in a proxy close to clients. Due to the large sizes of each media object and client viewing patterns that most of objects are only partially viewed [8], [9], *partial* proxy caching methods have been proposed, which divide each media object into smaller units, more

Manuscript received November 4, 2005; revised February 8, 2007. This work was supported by National Science Foundation Grants CNS-0405909, CNS-0509054/0509061, and CNS-0621629/0621631 and by a Grant from Hewlett-Packard Laboratories. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Simon Lucey.

S. Chen is with the Department of Computer Science, George Mason University, Fairfax, VA 22030 USA (e-mail: sqchen@cs.gmu.edu).

B. Shen and S. Wee are with the Mobile and Media System Lab, Hewlett-Packard Laboratory, Palo Alto, CA 94304 USA (e-mail: bo.shen@hp.com; susie.wee@hp.com).

X. Zhang is with the Department of Computer Science Engineering, Ohio State University, Columbus, OH 43210 USA (e-mail: zhang@cse.ohio-state.edu).

Digital Object Identifier 10.1109/TMM.2007.898943

1520-9210/$25.00 © 2007 IEEE

feasible for caching. There are two types of such methods. The first divides objects in the time domain [10]–[12], which we call *segment-based* approaches. The second is to divide objects in the media quality domain [13]–[16].

Although some algorithmic solutions and prototypes are available, today the practical usage and deployment of such systems are rare. Mocha [17] and QBIX [18] are prototype systems that divide media objects along the quality domain. Mocha is based on layered encoded streams, while QBIX tries to leverage MPEG4 and MPEG7 standards to do quality adaptation. However, they have not been widely deployed since they require extensive support from Internet Service Providers. For example, for Mocha, there are almost no layered-encoded streams provided online today. QBIX requires an online transcoding proxy, and does not work for videos in formats other than MPEG4 and MPEG7. Moreover, the quality of the media objects served in these systems is not controlled by the client, but by the service provider. Thus, they may not be client friendly.

When dividing media objects in the time domain (a segment-based approach), the aforementioned problems do not exist. The media with the original quality can always be served to the client. However, there are a number of technical problems.

First, multimedia objects are stored in container files, such as MP4 [19]. The file contains both audio and video tracks. In addition, it also contains indices to audio and video media packets, and may contain hint tracks with meta information. The flexibility of positioning these elements in the container file makes media-aware segmentation difficult for the proxy.

Second, media content is usually streamed using the RTP protocol, running on top of UDP. In practice, UDP traffic is likely to be blocked by firewalls at the client side due to security considerations. Also, Internet wide UDP-based communication raises reliability and fairness concerns. UDP packets are often subject to dropping at intermediate routers and switches. On the other hand, a large amount of unregulated UDP traffic unfairly throttles TCP traffic [20]. These concerns make it difficult to deploy the system based on UDP connecting the proxy and server [21].

Finally, after the object is segmented, the coordination between the caching of discrete object segments and the streaming of continuous media data is challenging. For example, although different online prefetching algorithms have been proposed to provide continuous streaming to clients, few measurement results in Internet streaming have been reported. Precise prefetching techniques [22] can provide continuous streaming with maximum resource utilization. However, system support is needed to accurately estimate the available bandwidth of the proxy–content-server link at runtime.

We have designed and implemented a segment-based proxy, named *SProxy*, to address these technical problems. It leverages existing Internet infrastructure and is able to serve and cache media objects in time-domain segments. As shown in Fig. 1, the deployment of *SProxy* does not require modifications on either the server side or the client side. This design takes advantage of the prevalence of HTTP, and eliminates most of the concerns about UDP based communications (especially when the proxy is placed inside the firewall).

The *SProxy* uses a segment-aware file I/O system that enables automatic segmentation and intelligent prefetching techniques to guarantee continuous streaming. This allows *SProxy* to transparently handle the complexity of media formats and to support continuous delivery demands. It has the following merits.

1) The *SProxy* handles client requests for streaming media objects via the standard RTSP [23] and RTP [24] protocols. It communicates with the content-server using the HTTP protocol. This design allows a regular Web server to serve streaming content, as well as regular Web documents. Thus, the existing Internet infrastructure is fully leveraged to address the flash crowd.
2) A client request is processed and divided into multiple subrequests. Each subrequest asks for only a small part of the whole media object. The sequence of subrequests is stopped whenever the client terminates its session, which subsequently terminates the data transfer. This design introduces a low startup latency while providing efficient bandwidth utilization.
3) Prefetching techniques are implemented to assist high quality continuous streaming. Based on dynamically detected available bandwidths of the proxy-server link, active prefetching techniques are used to dynamically prefetch the data likely to be accessed by the client.
4) The data contained in each segment is stored as a distinct object. The existing popularity based replacement policy is leveraged from the traditional Web proxy, and applied on these segments. It is a global, segment-based replacement policy instead of a media object-based one, which enables better utilization of the cache space.

An actual implementation of the *SProxy* is evaluated under various conditions. Our extensive experimental results show that *SProxy* consistently provides high quality streaming delivery to a medium number of concurrent clients, with reduced startup latency and more efficient cache utilization.

Compared with the commercial expensive Content Delivery/Distribution Networks (CDN) or Media Delivery Networks (MDN), SProxy provides an effective alternative to deliver high quality streaming media to end clients. Instead of fully duplicating each object to all dedicated edge servers, SProxy can easily leverage existing Internet proxies to deliver streaming data to clients on demand based on segment, which further reduces the bandwidth consumption. By enabling content providers to host media on a common Web server,

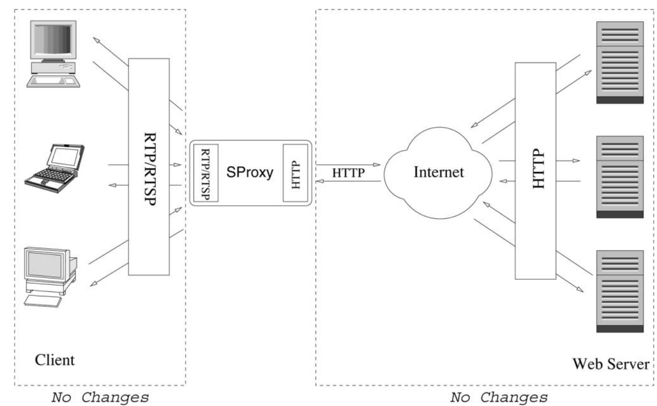

Fig. 1. Organization and protocols used in *SProxy*.

SProxy provides a novel infrastructure to facilitate the Internet streaming.

The rest of this paper is organized as follows. We review related work in Section II. We present the design and implementation of the *SProxy* in Section III. We evaluate the system performance through extensive experiments in Section IV. We make concluding remarks in Section V.

II. Related Work

The research on proxy caching of streaming media content has received much attention lately. Early efforts, e.g., Middleman [25], which has studied cluster of proxies for streaming media delivery, have considered little on one important feature of streaming media accessing. It is found that continuous media objects such as video or music clips are often partially accessed. Based on this observation, *partial* caching approaches have been proposed to reduce the cache space requirement. The basic strategy is to cache segments of objects that are divided in the viewing time domain. Typical examples include prefix caching [10], [26], [27], uniform segmentation [11], and exponential segmentation [12]. Prefix caching always caches the prefix of the objects to minimize the startup latency. In uniform segmentation, objects are cached in uniform-size segments, while in the exponential case, the segment size doubles along the viewing direction. Considering the limited resources available from a single cache, the Rcache [28] has considered the use of multiple proxies, focusing on the memory and disk utilization. These strategies focus on protocol design or benefit analysis based on synthetic workloads. Our work is based on the uniform segmentation caching strategy with the focus on real system implementations and evaluations of the system in real network environments using real workloads.

The partial caching strategy can be extended to the quality domain. Layered caching techniques [13], [14] have demonstrated efficient utilization of cache space by considering different QoS characteristics of client device or connectivity. A comparison with multiple version caching is studied in [15] while a model of layer-encoded object distribution is studied in [29]. In [30], the proposed approach attempts to select groups of consecutive frames by the selective caching algorithm, while in [31], the algorithm may select groups of non-consecutive frames for caching in the proxy. A different idea is proposed in video staging [32], in which a portion of bits from the video frames whose size is larger than a predetermined threshold is cut off and prefetched to the proxy a priori to reduce the bandwidth on the server proxy channel. Most of the partial caching schemes in the quality domain require layer-encoded objects or additional support from the proxy or client. The work presented in this paper does not have these requirements.

III. Design and Implementation of ***SProxy***

Fig. 2 shows the architecture of a *SProxy*, as well as its request handling. The *SProxy* is composed of four main components: a streaming engine that interfaces with the client, a segmentation-enabled cache engine that interfaces with content servers, a Local Content Manager and Scheduler (LCMS) module that coordinates the streaming engine and the cache engine, and a high speed disk that provides a fast data-path via the local file system.

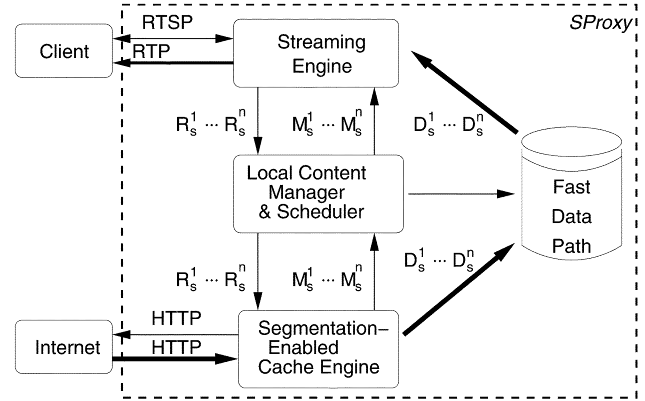

Fig. 2. Internal design of *SProxy* : A client request is divided into n subrequests with different ranges, R_s^1 to R_s^n, requesting different content segments, D_s^1 to D_s^n. LCMS controls when to send the next subrequest. The cache engine returns segment meta data (M_s^1 to M_s^n) to LCMS, and caches the segments D_s^1 to D_s^n on the disk.

A. Streaming Engine

The streaming engine is a multithreaded media server, responsible for providing an interface to the client. Its internal structure is described in detail in [33]. As shown in Fig. 2, it receives a client request for a RTSP URL and converts it to multiple segment requests, $R_s^1 \ldots R_s^n$, that are sent to LCMS. It uses the meta-data information, $M_s^1 \ldots M_s^n$, returned by the cache engine through LCMS to access the raw data segments on the disk.

As shown in Fig. 2, the streaming engine reads data segments, $D_s^1 \ldots D_s^n$, from the disk to serve clients. However, there is a problem: a randomly chosen segment length breaks the object into pieces, thus creating segments that are likely to include an incomplete media packet. If this incomplete packet is sent to the client, the client player would have to use error concealment or it may crash. One solution to this problem is to always segment the object on a packet boundary, which requires the *SProxy* to have packet boundary knowledge *before* segmentation can be done. This information could be obtained by parsing the complete media file, or by using a hint track, if available. However, the hint track data can be dispersed through the media file, so in either case, the whole file may have to be downloaded. A better solution is to allow random segment boundaries, but to always feed a complete data packet to the client. In the *SProxy*, a segment-aware file I/O system is implemented to support this requirement. It automatically requests the appropriate segment when reading or seeking beyond the boundaries of the current segment. LCMS tries to ensure that the next segment is always available in the cache.

B. Local Content Manager and Scheduler

Local Content Manager and Scheduler (LCMS) coordinates the streaming engine and the segmentation-enabled cache engine. It converts the subrequests, e.g., $R_s^1 \ldots R_s^n$, to corresponding HTTP requests (with Range headers) and forwards them to the proxy. It returns the appropriate cache meta-data

$M_s^1 \ldots M_s^n$ from the proxy replies to the streaming engine. More importantly, LCMS schedules segment prefetching. Prefetching is necessary because segment-based proxy caching is a partial caching solution, in which only a part of the object is cached in the proxy while a client may access an object to a segment which is not cached in the system. To guarantee continuous media delivery, each segment should be available locally before the streaming engine tries to read and stream to the client. Otherwise, the client can experience playback jitter.

Based on the available bandwidth, to prefetch the uncached segment at a proper time can not only maintain the continuous service, but also reduce resource waste since the client may terminate any time without viewing all the prefetched data.

We have implemented multiple segment based caching modes and provided analytical models in [22]. In the following context, we briefly describe the four modes we implemented in our streaming proxy, depending on when the request for a succeeding uncached segment is issued.

- *OnDemand*: In this mode, no prefetching is implemented. The succeeding segment is fetched when it is needed by the streaming engine. This mode is simple and works fine when the available bandwidth of HTTP channel is large enough. Otherwise, streaming can be interrupted due to the delay in fetching the next segment from the server. Some of these effects can be partially hidden by providing buffering in the streaming engine.
- *Window*: In this mode, the subrequest for the next uncached segment is always issued when the client starts to access the current one. Thus it provides aggressive prefetching with a look-ahead window size of one segment.
- *Half*: Intuitively, the window size is adjustable. We also implemented a *Half* mode, in which the subrequest for the next uncached segment is issued after the server has reached the middle of the current one. Thus, in this mode, the window size is half of a segment length.
- *Active*: Active prefetching is implemented to dynamically decide when to prefetch an uncached segment according to the real-time bandwidth. It is the most precise online prefetching technique according to [22] and is implemented with the aid of Packet CAPture (PCAP) library [34]. With the API provided by PCAP, we periodically estimate the available network bandwidth between the *SProxy* and the content-server. The prefetch schedule is then computed using the media encoding rate extracted from the header of the media file, which corresponds to the desired data transmission rate between *SProxy* and the client.

C. Segmentation-Enabled Cache Engine

The segmentation-enabled cache engine handles the subrequests from LCMS. In case of a cache MISS, the cache engine gets the data for the subrequest from the content-server (or other peering proxies). The cache stores data D_s^n (data for segment n) on the disk, as well as constructing and sending a reply with meta data M_s^n only to LCMS. The meta data includes the name and the location of the file containing the data for this subrequest on the local disk. In a case of a cache HIT, the cache directly constructs and sends the M_s^n meta-data to LCMS.

Currently, *SProxy* uses a modified version of Squid2.3 (STABLE4) as the cache engine. Segmentation support is provided through the Range header in HTTP requests. Squid identifies objects in its cache using the MD5 hash of the request URL. Hence, in the original version of Squid, different ranges of a URL would have the same MD5 keys, and HTTP requests that include the Range header would be considered non-cachable. To make these requests cachable, our segmentation-enabled version rewrites the URL internally to enable the caching of different segments of a media object. The rewritten URL is used internally in the proxy to identify different range requests. If the corresponding segment is not cached, the request is forwarded to the content-server (or peering proxies) by restoring back the URL and Range header.

Since the rewriting of the URL provides the opportunity to cache the data for different segments of the same object, segment caching is enforced by saving partial data on disk without violating the HTTP protocol. In the implementation, a HTTP reply status of PARTIAL CONTENT (206) indicates the reply corresponds to a range request.

Popularity based replacement policy has been found to be the most efficient for multimedia object caching. *SProxy* leverages the existing popularity based replacement policy in Squid. In our system, it is not a pure popularity based replacement due to the LOCK problem when streaming. We will discuss more about it later.

Additionally, cooperative proxies have been used for caching static Web objects. It is even more desirable for caching large streaming media objects. *SProxy* also leverages the existing cooperative functions in Squid. When requesting segments from neighboring caches, the internally re-written URL is restored to the original version, with the Range header added. This allows *SProxy* interact with regular Web-proxies without streaming capability, as well as other streaming-enabled Squid proxies. The procedure is as follows: after a request gets a "miss" from its local cache, its neighbor proxy cache is searched if any by changing the internal request to the one as we showed before. We omit the details here.

D. Fast Data Path

The shared local file-system provides a fast data path between segmentation-enabled cache engine and the streaming engine. Traditionally, Squid transfers incoming data to an HTTP client over a network. For large media data files, it is more efficient to directly share the part of file system used as a data cache by Squid. In the *SProxy* system, a set of new methods, PREFETCH, LOCATEFILE and LOCK, are added to Squid for this purpose.

1) PREFETCH is implemented as a nonblocking version of the HTTP GET method. Whenever a segment is required, a request with a PREFETCH method and the corresponding Range header is sent to the proxy. The proxy checks if the requested segment is cached or not. If it is cached, a HIT is returned. Otherwise, a MISS is returned and the corresponding request is re-written as a HTTP GET and forwarded to the content-server or peer simultaneously. The proxy will store the reply containing the requested segment data on its local disk for future requests.

2) LOCATEFILE is implemented as a blocking method. LCMS only invokes this method after a PREFETCH request returns a HIT. It returns the file location of the requested segment in the cache file structure maintained by Squid. It blocks until the entire data for a range request has been written to disk.
3) LOCK is used before the streaming engine starts to stream a segment to the client. Since the segment is cached and the cache is managed by Squid, the replacement policy in Squid automatically starts the replacement when the available cache space is below some threshold. It does not know whether or not the to-be-replaced segment is being used by the streaming engine. Thus, before reading the data of a segment for streaming, LCMS issues a request with a LOCK method. This ensures that the to-be-read file will not be a candidate for eviction. After segment access is complete, the lock is released.

The nonblocking PREFETCH method and the blocking LOCATEFILE method effectively split the original, blocking GET method into a two-phase protocol. This is critical to the system performance when the *SProxy* needs to handle a large number of concurrent requests or when the segment size is large. Multiple PREFETCH methods for different segments can be issued without locking up the LCMS. The design of LOCK provides a tool to coordinate the streaming engine and cache engine.

IV. Performance Evaluation

In this section, we describe the test setup and evaluation metrics that we use in experiments. We then present detailed experimental results upon multiple concurrent requests, including a full caching approach to provide baseline values. Four case studies of *SProxy* are presented then.

A. Test Setup

We run tests in real network settings using actual implementation of the content-server, *SProxy*, and media clients. We use Apache Web Server (version 2.0.45 with HTTP 1.1) as the content-server. It is hosted on an HP Netserver lp1000r, with a 1-GHz Pentium III Linux PC platform. The *SProxy* system runs on a HP workstation x4000 with two dual 2-GHz Pentium III Xeon Linux PC, with 1-GB memory. The media client used for the experiments is a dummy loader that logs incoming RTP and RTSP packets.

For all tests, the network connection between *SProxy* and the client machine is a switched 100-Mbps Ethernet. For network conditions to the content-server, three setting are used, namely *local, remote* and *controlled* environments. The local environment is set up with both the content-server and the *SProxy* system connected via a switched 100-Mbps Ethernet within HP Labs (Palo Alto, CA USA). The remote environment is constructed with the *SProxy* system and the content-server at trans-Pacific sites (the U.S. and Japan, respectively). The bottleneck bandwidth is approximately 10 Mbps. To study the effectiveness of four prefetching methods in different network settings, we also construct a controlled environment in which the link capacity between the proxy and the content- server can vary. We use traffic control support in the Linux kernel via the *tc(8)* utility to the bottleneck bandwidths link.

B. Evaluation Metrics

We evaluate two end-to-end statistics that are especially relevant to streaming media delivery, client perceived startup latency and client perceived jitter. Startup latency is measured as the interval between the client sending a request for a media stream (the RTSP DESCRIBE method), and the arrival of the first media packet. The client jitter is the average of the values in the Receiver Report RTCP messages, calculated based on the algorithm in Appendix A.8 of RFC 1889 [24]. It represents the statistical variance of the media packet inter-arrival time.

To better evaluate different prefetching methods, we also record two statistics specific to *SProxy*. We instrument the proxy system to measure time spent in handling each segment request. It is measured as the interval between the time when a segment is requested and the time the location of the segment in the cache storage is returned to the proxy. Note that every request for a segment results in a Squid handshake (to check whether it is in cache), while an uncached segment causes an HTTP transfer from the content-server. This measurement reveals whether the proxy can fetch segments in time in the middle of streaming sessions.

C. Experimental Results

We first perform experiments using a full object caching approach. The results provide a basis for comparisons with our segment-based approach. Furthermore, to study the performance of our *SProxy*, we conduct experiments in four different aspects. We first consider the effects of using different segment sizes in the *SProxy*, when the number of concurrent clients increases. Then we evaluate the performance difference when the content-server sits at different network distances from the *SProxy*. We further evaluate the effectiveness of each prefetching method under different *SProxy*—content-server link bandwidth capacities. For each of these experiments, the cache size is set large enough to store all the fetched content, and each client accesses a unique object. Moreover, the clients play clips in their entirety. Since there is no segment re-use across clients, this represents the worst case behavior for a cache engine. We finally validate our results with a trace driven simulation, using real enterprise access patterns. These traces include multiple clients accessing the same clip, and clients that do early termination.

1) Full Caching Approach: In the full caching approach, media objects are not segmented, but fetched in their entirety. In this experiment the client, *SProxy* and content- server are located in the same local network. Experiments are performed on different video clips of length 1, 2, 5, 10, 20, 40 min, encoded at 112 kbps. The cache size is set large enough so that there are no capacity misses, hence no replacement is necessary. The results are averaged over ten runs.

Fig. 3(a) shows that the startup latency perceived by the client, as expected, increases linearly with the video size. Similar trends are reflected in performance in terms of the average time to handle a miss as shown in Fig. 3(b). Fig. 3(c) shows that the handshake time in the proxy for the full caching approach also increases linearly with the video size. Note that it is substantially smaller than the corresponding miss process time. In the full proxy caching approach, each media object

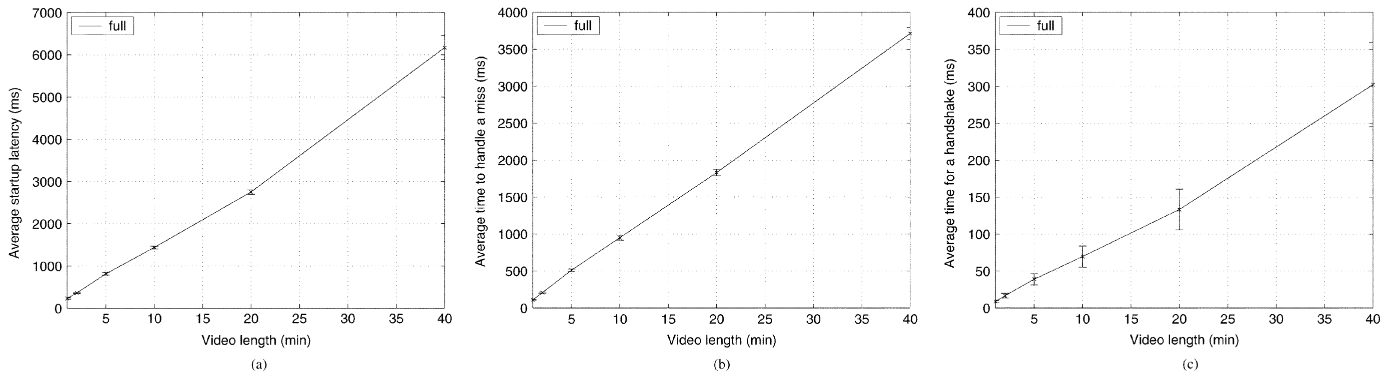

Fig. 3. Performance of the full caching approach.

is only fetched once from the content-server at the beginning, after that all requests hit in the proxy, thus the client-perceived jitter is very small, negligible in this situation.

2) Effect of Segment Size: In the previous section, where each object is accessed and stored in entirety, we have learned that the performance of the full caching approach depends on the media object length and degrades almost linearly with the increase of the object length. In *SProxy*, objects are segmented and managed as smaller units. The next experiment tests the effect of segment size on the *SProxy* performance. In this set of experiments, an increasing number of concurrent clients request unique media objects. The media objects are all copies of the same piece of content (a 2-min video clip) with different names. This effectively disables the file buffer cache in the Operating System. Moreover, the media data served to each client is identical, which allows us to present the data as averages across each client session. The client request inter-arrival interval is . The Squid cache file system is re-initialized before each experiment. We evaluate the performance by running tests with *SProxy* using different segment sizes for segment-based caching. These experiments are carried out in the local environment using the *OnDemand* mode. Thus, there is no explicit prefetching, so we can isolate the effect of the segment size.

Fig. 4(a) shows the client perceived startup latency when the segment size varies from 100 to 500 KB and when the segment size is large enough to include the entire object. Clearly, the startup latency increases when the base segment size increases since *SProxy* waits until the first segment is fetched from the content-server before starting streaming to the client. It is also expected that the startup latency increases when the number of concurrent clients increases, since this puts a load on the streaming server. Compared to the startup latency when the entire object is fetched as one segment, the startup latency in *SProxy* is only about 30% to 60%. It is also found that when the segment size increases beyond 300 KB, the client perceived startup latency increases faster, while the effect is less pronounced when the segment size varies in the range of 100 to 300 KB. The startup latency is proportionally larger for clients in the remote environment, and not shown for brevity.

Fig. 4(b) shows the client perceived jitter. It is obvious that jitter is the smallest when the segment size is large enough to include the entire object. Otherwise, additional jitter may be perceived due to the on-demand nature of segment based fetching by *SProxy*. We show in Fig. 4(c) and (d) that the average time consumed for the proxy to handle a MISS and a Squid handshake, respectively. It is clear that the average consumed time to handle both a MISS and a handshake increases with the segment size. Comparing Fig. 4(a) and (c), we note that the client perceived startup latency is usually larger than the time to handle a cache MISS. This is because the startup latency includes the time to setup the streaming session in addition to the time of fetching the first segment.

This set of experiments shows that *SProxy* outperforms the full caching approach in terms of the client perceived startup latency and the average time to handle a miss, while it causes a comparable amount of playback jitter even without prefetching support.

3) Effect of Proximity: Another factor that affects the scalability of *SProxy* is the proximity: the distance between the content-server and *SProxy*. We evaluate the performance by running tests with the content-server located in the local environment as well as in the remote environment. A segment size of 100 KB is used for this set of experiments. For fairness we run the *Local-Ondemand* again with other segment sizes. Its results slightly differ from those in the previous subsection.

Fig. 5(a) shows the startup latency for local accesses, while Fig. 5(b) shows this metric for remote accesses. In the local case, it varies from 96 to 169 ms, while for remote accesses, the startup latency is much larger, with a much bigger dynamic range, from 2 to 11 s. The startup latency in both environments shows only a small variation across different prefetching methods. This is an intuitive result, since the value would be dominated by the access time for the *first* segment accessed. It is also seen that the startup latency generally increases when there are more concurrent requests. These results indicate that more concurrent requests can be served in local networks, and that more concurrent requests can lead to a longer startup latency in wide area networks. This figure also shows that our design and implementation of *SProxy* can support the delivery of media objects with reasonable startup latency in both intranet and Internet environments.

Another important aspect of streaming media delivery is whether the proxy can provide rigorous continuous streaming. Fig. 6(a) and (b) show the average time that the proxy consumes to process a cache MISS in each environment. The average time to handle a MISS in the local testing environment is less than 23 ms. In a wide-area network, the average consumed time can reach 6.5 s. The results justify that prefetching for content from

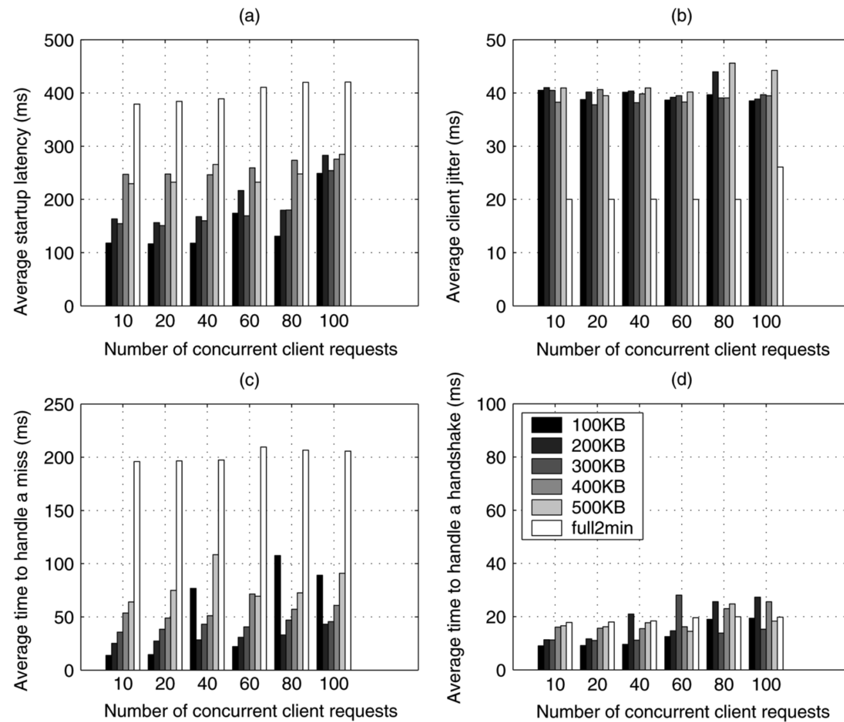

Fig. 4. Performance study with different segment sizes.

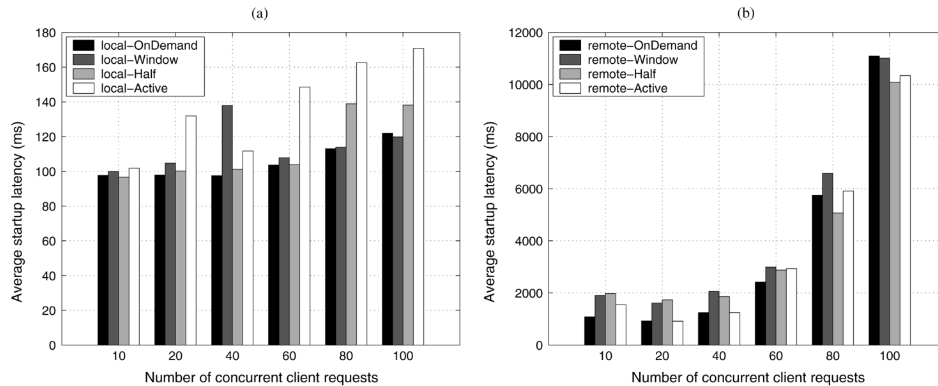

Fig. 5. Client startup latency for local and remote.

remote content servers is necessary, since such a large delay may potentially cause playback jitter at the client side.

Fig. 7(a) and (b) show the average time for the proxy to handle a Squid handshake, whether a HIT or a MISS, for the local and remote environment, respectively. A good prefetching method will have a higher percentage of HIT cases, which leads to a correspondingly smaller average time. Comparing Fig. 7 with Fig. 6, we can see that a handshake consumes much less time than a MISS on average. Note that *OnDemand* does not do any prefetching. It shows some HIT cases, since the file format parser might request the same segment multiple times, e.g., first for parsing the hint track, and then for reading media data. As shown in the remote case of Fig. 7, *OnDemand* always consumes more Squid handshake time and other prefetching methods reduce the Squid handshake time somewhat. It seems that a simple *Window* mode performs the best in this set of tests. We have shown in [35] that *Active* should perform best if an accurate real-time measurement of the proxy-server link bandwidth is in place. Our current implementation of *Active* may have been limited by PCAP's capability.

Fig. 8(a) and (b) show the client perceived jitter in both local and remote environments. In both cases, the absolute client perceived jitter is small, which indicates that our *SProxy* can successfully serve a large number of clients with rigorous continuous streaming demand. Note that the client jitter tends to increase when more concurrent requests are served, especially in the remote environment. This indicates that accurate prefetching is very important especially when the *SProxy*—content-server link bandwidth resource becomes scarce. *Active* prefetching achieves better performance as shown in the remote case.

4) Prefetching Effectiveness: The preceding experiments have evaluated the system performance in local and remote network settings. To further study the effectiveness of the different prefetching methods in different network settings, we test the system in a controlled environment, as described in

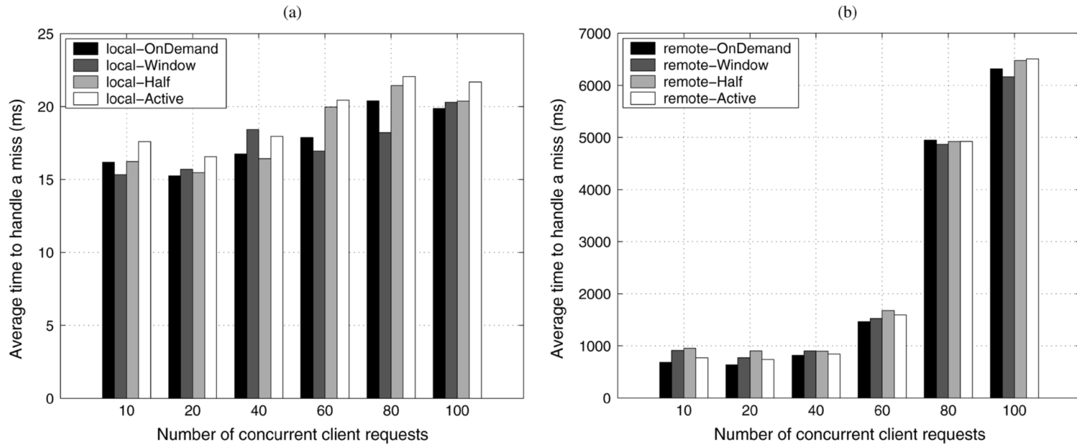

Fig. 6. Time to handle a MISS for local and remote.

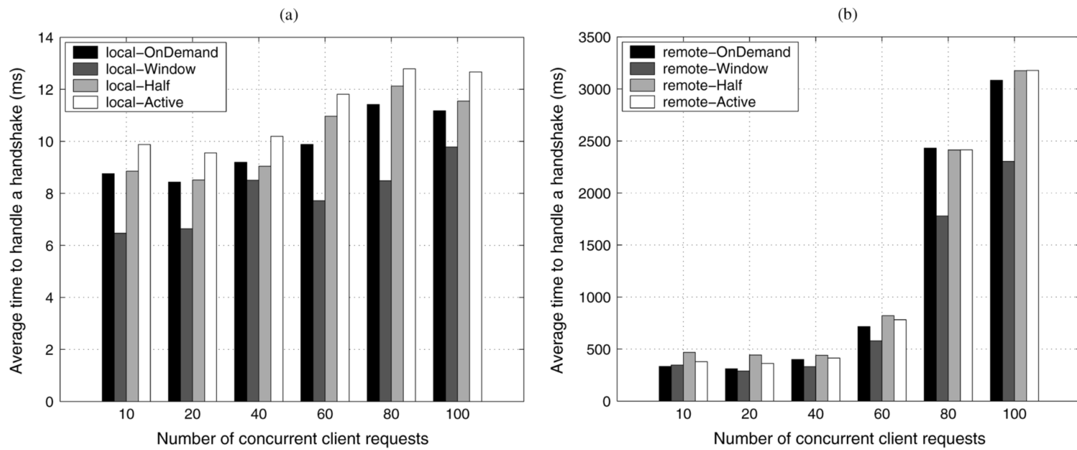

Fig. 7. Time to handle a handshake for local and remote.

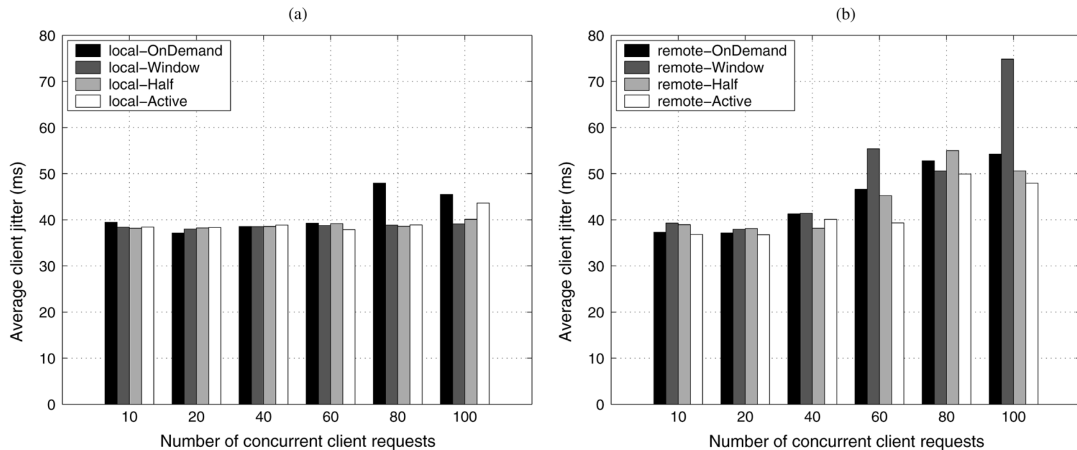

Fig. 8. Client-perceived jitter for local and remote.

Section IV-A. For each bandwidth setting, a video clip with an encoding rate of 75 Kbps is served from the content-server to the client through *SProxy*. We collect the Squid handshake time and client jitter statistics for each prefetching method. Please note that in all prefetching methods, *Active* schedules prefetching based on the detected available bandwidth, called bottleneck bandwidth, which we use *tc* to control during the experiments.

As shown in Fig. 9(a) and (b), both the Squid handshake time and the client perceived jitter decrease when the bottleneck bandwidth increases. Note that the Squid handshake time here is generally much longer than the result in the proximity study since the bottleneck link bandwidth is much smaller. The *Active* prefetching can be seen to have the shortest Squid handshake time, especially when the link bandwidth is low. The differences in client-perceived jitter are less obvious although *Active* and *Window* methods perform better with a low bottleneck link bandwidth. A notable average client jitter increase happens on Fig. 9(b) when the bottleneck bandwidth changes from 400

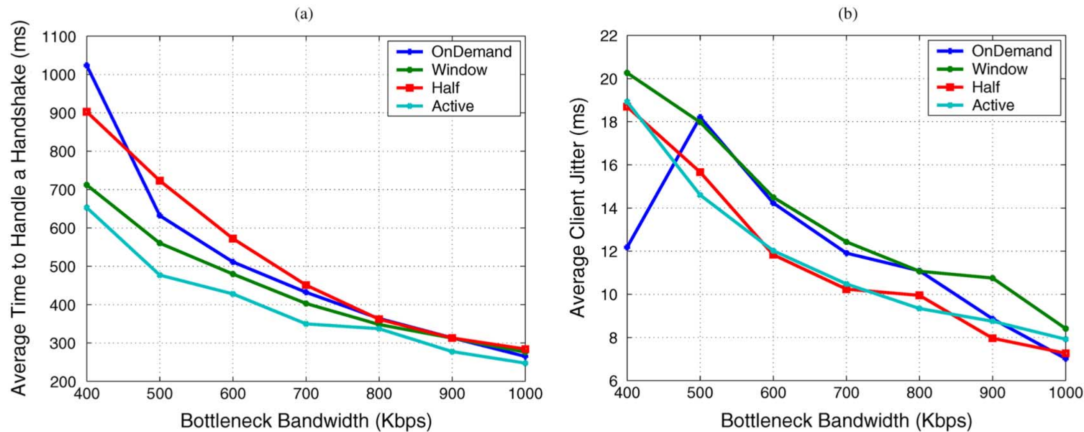

Fig. 9. Squid handshake time and client perceived jitter.

to 500 Kbps. Our analysis of the raw data indicates that a much smaller jitter value is captured when the bottleneck bandwidth is set to 400 kbps by *tc* for the first time.

5) Cache Efficiency Study Using Real Workload: Even though we use segmentation-enabled Squid as the cache engine in our *SProxy*, it is important to evaluate its performance on cache efficiency in conjunction with prefetching methods. To this purpose, we use a trace extracted from real enterprise media server logs to drive a 24-h run of the actual system in the local environment. These traces include clients that access the same clip, and clients that terminate a clip prematurely, or start playing a clip from the middle. Thus we would expect better caching behavior, but also wasted bandwidth due to segments that are pre-fetched and never used.

The trace contains 16238 requests with the access duration varies from 1 to 50 minutes. In these 16 238 requests, 92% of them are demanding a same video clip, most of which are with premature terminations. Thus, the caching performance is expected to be very high. We select such a workload because we also want to test whether the system can survive a large number of concurrent requests in a long time period. There is a total of 70.775 GB data accessed. The unique object size amounts to 5.358 GB. Based on the file length and streaming rate of the objects requested, we have created matching video clips in MP4 file format. A content pool is created as follows using the parameters as shown in Table I.

As shown in the table, video objects are created with six bitrate (28, 56, 112, 156, 180, 256 kbps) versions with a file length of 1, 2, 5, 10, 20, 50, 100 min. The request duration of each access is extracted from the trace.

Based on this real enterprise media access trace, we use a cache simulator to evaluate both full- and segment-based caching strategies. We also use the real system runs to verify the simulation-based results of segment based caching strategies. We use simulations since some metrics, such as false prefetch we shall see soon, are very difficult to measure in the real runs. For segment-based caching strategies, we also evaluate various prefetching methods and different segment sizes.

Fig. 10(a) compares the total server traffic amount generated by the full and various segment-based caching strategies based on simulations. Note that segment-based caching generates significantly less server traffic. Server traffic is very low even

TABLE I
CONTENT AND ACCESS PARAMETERS OF REAL WORKLOAD

Rate (Kbps)	File length (minute)	Max access duration (minute)
28	1, 10, 20, 50	1
56	50	12
112	1,2,5,10,20,50	14
156	1,20,50	14
180	2,5,10,20,50,100	50
256	1,2,5,10,20,50,100	25

with a small cache space. We have validated this by driving the same workload through actual runs using *SProxy*. Looking into server traffic produced by segment-based caching with different prefetching methods, Fig. 10(b) shows that *OnDemand* generates the least server traffic since it does not do any prefetch; *Half* and *Window* methods, with increased aggressiveness in prefetching, generate more and more server traffic. This is a small penalty *SProxy* pays to improve continuous streaming of media content. *Active* prefetching is not simulated since it is difficult to simulate a dynamic channel bandwidth between the content-server and *SProxy*. The server traffic amount generated by *Active* prefetching would depend on the time-varying nature of the channel bandwidth. However, not all prefetched segments will be used by the clients. We define false prefetch as the size of the segments that are prefetched and cached but have never been streamed to clients before they are evicted. Fig. 11 shows for this trace, *Half* method produces about 50% of false prefetches compared to *Window*. Thus, for real traces, the *Window* method is too aggressive, since many clients terminate playing early (i.e., before accessing half of a segment). Also, since the prefetching granularity is segment, smaller segment size produces less false prefetch.

V. CONCLUSION

Recent years have seen a lot of streaming service un-availability due to flash crowds and a large amount of research work

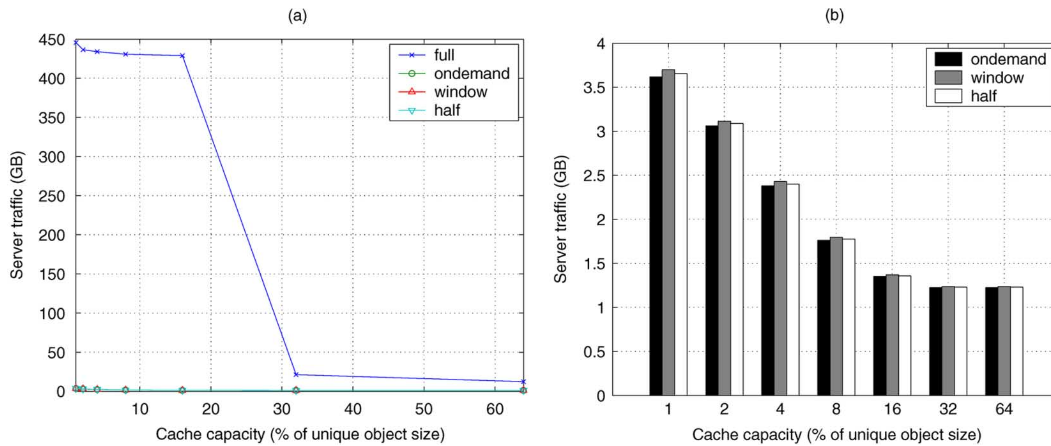

Fig. 10. Server traffic for full and segment-based caching.

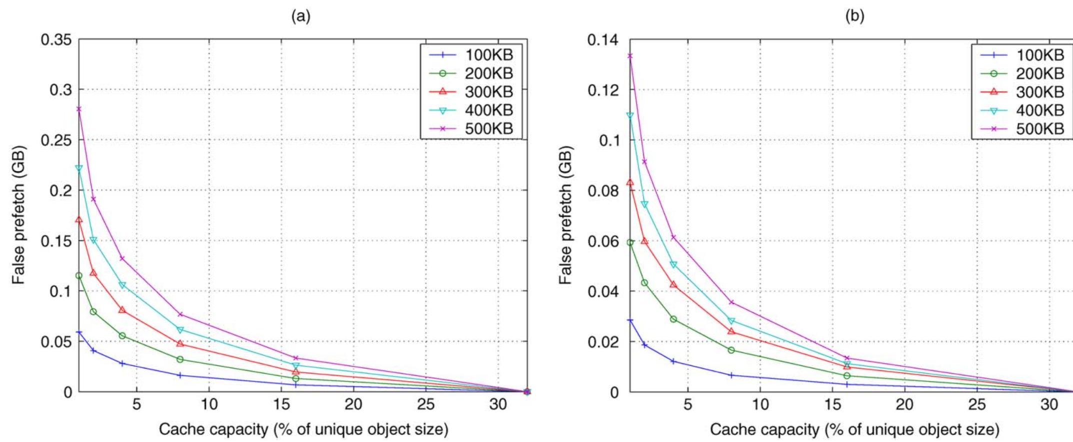

Fig. 11. False prefetch by *Window* and *Half*.

in segment-based proxy caching to deal with this issue. However, its implementation and deployment are hindered by several factors. Additionally, system support is demanded to guarantee continuous streaming. Through our design, implementation, and testing of such a system, we conclude that it is possible to push the streaming capability to the edge of the network and couple it with a caching proxy to efficiently serve a large number of clients. This design fundamentally frees the content provider from serving constraints. Specifically, our contributions are the following.

- We have designed and implemented a segment-based caching proxy that can successfully shoulder the server's burden and support concurrent streaming of multimedia content to a medium number of clients with rigorous latency and continuity constraints.
- The design and implementation leverage the existing Internet infrastructure. The content-server needs only to be a simple Web server, yet its contents are served through *SProxy* in a scalable and efficient fashion.
- We have thoroughly evaluated different prefetching methods which are closely coupled with the segment-based caching. We have shown that segment-based access inherently reduces the client perceived startup latency and various prefetching methods can provide continuous streaming in various network conditions.
- We have tested the full system within real network conditions and with a real workload. We believe this is the first work of this kind.

Currently, the *SProxy* system is deployed at many sites of a large enterprise for practical usage and at a site for scalability tests.

ACKNOWLEDGMENT

The authors appreciate the critical and constructive comments from the anonymous referees.

REFERENCES

[1] Inktomi, Streaming Media Caching White Paper Inktomi Corporation, 1999 [Online]. Available: http://www.inktomi.com/products/traffic/streaming.html, in Technical Report

[2] G. Gibson, J. Vitter, and J. Wilkes, "Storage and i/o issues in large-scale computing," in *ACM Workshop on Strategic Directions in Computing Research*, 1996 [Online]. Available: http://www.medg.lcs.mit.edu/doyle/sdcr

[3] S. Saroiu, K. P. Gummadi, R. J. Dunn, S. D. Cribble, and H. M. Levy, "An analysis of Internet content delivery systems," in *Proc. 5th Symp. Operating Systems Design and Implementation (OSD1)*, Boston, MA, Dec. 2002.

[4] A. Wolman, G. Voelker, N. Sharma, N. Cardwell, M. Brown, T. Landray, D. Pirinel, A. Karlin, and H. Levy, "Organization-based analysis of Web-object sharing and caching," in *Proc. 2nd USENIX Symp. Internet Technologies and Systems (US/TS)*, Boulder, CO, Oct. 1999.

[5] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Cribble, H. M. Levy, and J. Zahorjan, "Measurement, modeling and analysis of a peer-to-peer file-sharing workload," in *Proc. 19th ACM Symp. Operating Systems Principles (SOSP)*, Bolton Landing, NY, Oct. 2003.
[6] L. Guo, S. Chen, Z. Xiao, and X. Zhang, "Analysis of multimedia workloads with implications for internet streaming media," in *Proc. 14th Int. World Wide Web Conf.*, Chiba, Japan, May 2005.
[7] V. N. Padmanabhan, H. J. Wang, and P. A. Chou, "Resilient peer-to-peer streaming," in *Proc. IEEE Int. Conf. Network Protocols (ICNP)*, Atlanta, GA, Nov. 2003.
[8] L. Cherkasova and M. Gupta, "Characterizing locality, evolution, and life span of accesses in enterprise media server workloads," in *Proc. ACM NOSSDAV*, Miami, FL, May 2002.
[9] M. Chesire, A. Wolman, G. Voelker, and H. Levy, "Measurement and analysis of a streaming media workload," in *Proc. 3rd USENIX Symp. Internet Technologies Arid Systems*, San Francisco, CA, Mar. 2001.
[10] S. Sen, J. Rexford, and D. Towsley, "Proxy prefix caching for multimedia streams," in *Proc. IEEE INFOCOM*, New York, NY, Mar. 1999.
[11] R. Rejaie, M. Handley, H. Yu, and D. Estrin, "Proxy caching mechanism for multimedia playback streams in the internet," in *Proc. Int. Web Caching Workshop*, San Diego, CA, Mar. 1999.
[12] K. Wu, P. S. Yu, and J. Wolf, "Segment-based proxy caching of multimedia streams," in *Proc. WWW*, Hong Kong, China, May 2001.
[13] R. Rejaie, M. H. H. Yu, and D. Estrin, "Multimedia proxy caching mechanism for quality adaptive streaming applications in the internet," in *Proc. IEEE INFOCOM*, Tel-Aviv, Israel, Mar. 2000.
[14] R. Rejaie, M. Handely, and D. Estrin, "Quality adaptation for congestion controlled video playback over the internet," in *Proc. ACM SIGCOMM*, Cambridge, MA, Sep. 1999.
[15] T. Kim and M. H. Ammar, "A comparison of layering and stream replication video multicast schemes," in *Proc. ACM NOSSDAV 2001*, Port Jefferson, NY, Jun. 2001.
[16] P. D. Cuetos, D. Saparilla, and K. W. Ross, "Adaptive streaming of stored video in a tcp-friendly context: Multiple versions or multiple layers?," in *Proc. Packet Video Workshop*, Apr. 2001.
[17] R. Rejaie and J. Kangasharju, "Mocha: A quality adaptive multimedia proxy cache for internet streaming," in *Proc. ACM NOSSDAV*, Port Jefferson, NY, Jun. 2001.
[18] P. Schojer, L. Boszormenyi, H. Hellwagner, B. Penz, and S. Podlipnig, "Architecture of a quality based intelligient proxy (qbix) for mpeg-4 videos," in *Proc. WWW*, Budapest, Hungary, May 2003.
[19] R. Koenen*, Overview of the Mpeg-4 Version 1 Standard* Mar. 2001 [Online]. Available: http://wwwam.hhi.de/mpeg-video/standards/mpeg-4.htm
[20] L. Huang, U. Horn, F. Hartung, and M. Kampmann*, Proxy-Based Tcp-Friendly Streaming Over Mobile Networks*. Atlanta, GA: , Sep. 2002.
[21] E. Bommaiah, K. Quo, M. Hofmann, and S. Paul, "Design and implementation of a caching system for streaming media over the internet," in *Proc. IEEE RTAS*, Washington, DC, May 2000.
[22] S. Chen, B. Shen, S. Wee, and X. Zhang, "Streaming flow analyses for prefetching in segment-based proxy caching strategies to improve media delivery quality," in *Proc. 8th Int. Workshop on Web Content Caching and Distribution*, Hawthorne, NY, Sep. 2003.
[23] H. Schulzrinne, A. Rao, and R. Lanphier, Real Time Streaming Protocol (RTSP) Apr. 1998, RFC2326.
[24] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, RTP: A Transport Protocol for Real-Time Applications Jan. 25, 1996, RFC 1889.
[25] S. Acharya and B. Smith, "Middleman: A video caching proxy server," in *Proc. ACM NOSSDAV*, Chapel Hill, NC, Jun. 2000.
[26] S. Gruber, J. Rexford, and A. Basso, "Protocol considertations for a prefix-caching for multimedia streams," *Comput. Network*, vol. 33, no. 1-6, pp. 657–668, Jun. 2000.
[27] B. Wang, S. Sen, M. Adler, and D. Towsley, "Proxy-based distribution of streaming video over unicast/multicast connections," in *Proc. IEEE 1NFOCOM*, New York City, NY, Jun. 2002.
[28] Y. Chae, K. Quo, M. Buddhikot, S. Suri, and E. Zegura, "Silo, rainbow, and caching token: Schemes for scalable fault tolerant stream caching," *IEEE J. Select. Areas Commun.*, vol. 20, no. 7, pp. 1328–1344, Sep. 2002.
[29] J. Kangasharju, F. Hartanto, M. Reisslein, and K. W. Ross, "Distributing layered encoded video through caches," in *Proc. IEEE INFOCOM*, Anchorage, AK, Apr. 2001.
[30] W. Ma and H. Du, "Reducing bandwidth requirement for delivering video over wide area networks with proxy server," in *Proc. IEEE ICME*, 2000, vol. 2, pp. 991–994.
[31] Z. Miao and A. Ortega, "Scalable proxy caching of video under storage constraints," *IEEE J. Select. Areas Commun.*, vol. 20, no. 7, pp. 1315–1327, Sep. 2002.
[32] Z. 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 Trans. Netw.*, vol. 8, no. 4, pp. 429–442, Aug. 2000.
[33] S. Roy, J. Ankcorn, and S. Wee, "Architecture of a modular streaming media server for content delivery networks," in *Proc. IEEE Int. Conf. Multimedia & Expo*, Baltimore, MD, Jul. 2003.
[34] [Online]. Available: http://www.tcpdump.org/PCAP
[35] S. Chen, B. Shen, S. Wee, and X. Zhang, "Designs of high quality streaming proxy systems," in *Proc. IEEE INFOCOM*, Hong Kong, China, Mar. 2004.

Songqing Chen (M'03) received the Ph.D. degree in computer science from the College of William and Mary, Williamsburg, VA.

He is an Assistant Professor of Computer Science at George Mason University, Fairfax, VA. His research interests include Internet content delivery systems, Internet measurement and modeling, operating systems and system security, and distributed systems and high-performance computing. He can be reached via sqchen@cs.gmu.edu.

Bo Shen (M'97–SM'04) received the B.S. degree in computer science from Nanjing University of Aeronautics and Astronautics, Nanjing, China, and the Ph.D. degree in computer science from Wayne State University, Detroit MI.

He is a Senior Research Scientist in Streaming Media Systems Group at Hewlett-Packard Laboratories. His research interests include multimedia signal processing, multimedia networking and content distribution systems. He has published over 50 papers in prestigious technical journals and conferences, and holds six U.S. patents, with many pending. He has also been on the Program Committee for a number of technical conferences, including ACM Multimedia and IEEE ICDCS. He is currently an Associate Editor of the IEEE TRANSACTIONS ON MULTIMEDIA.

Susie Wee received the Ph.D. degree in electrical engineering from the Massachusetts Institute of Technology, Cambridge.

She is currently the Laboratory Director of the Mobile and Media Systems Laboratory at Hewlett Packard Laboratories, Palo Alto, CA, and is a Consulting Assistant Professor at Stanford University, Stanford, CA. Her research interests include multimedia networking and secure streaming.

Dr. Wee received the Technology Review's Top 100 Young Investigators Award in 2002. She is currently an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS, SYSTEMS, AND VIDEO TECHNOLOGY and formerly served as an Associate Editor of the IEEE TRANSACTIONS ON IMAGE PROCESSING.

Xiaodong Zhang (SM'94) received the B.S. degree in electrical engineering from Beijing Polytechnic University, Beijing, China, and the Ph.D. degree in computer science from the University of Colorado at Boulder.

He is the Robert M. Critchfield Professor of Engineering and Chair of Department of Computer Science and Engineering at The Ohio State University, Columbus. He served as the Program Director of Advanced Computational Research at the National Science Foundation during 2001–2004.

Dr. Zhang is the Associate Editor-in-Chief of the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS and is also serving on the Editorial Boards of the IEEE TRANSACTIONS ON COMPUTERS and *IEEE Micro*.