A CASE FOR INTERNET STREAMING VIA WEB SERVERS

Songqing Chen[1], Bo Shen[2], Wai-tian Tan[2], Susie Wee[2], and Xiaodong Zhang[3]

[1] Dept. of Computer Science, George Mason University, Fairfax, VA 22030
[2] Mobile & Media Systems Lab, HP Laboratories, Palo Alto, CA 94304
[3] Dept. of Computer Science and Engineering, Ohio State University, Columbus, OH 43210

ABSTRACT

Hosting Internet streaming services has its unique challenges. Aiming at making Internet streaming services be widely and easily adopted in practice, in this paper, we have designed and implemented a system, called *SProxy* that can leverage existing Internet infrastructure to free the streaming content providers so that they only need to host streaming content through a regular Web server. *SProxy* has been extensively tested and evaluated and it provides high quality streaming delivery in both local area networks and wide area networks (e.g. between Japan and US).

1. BACKGROUND AND MOTIVATION

With the increase of aggregate Internet bandwidth, the demand of streaming media applications, such as tele-education, tele-medicine, entertainment grows quickly. However, the delivery of diverse streaming media contents on IP networks in a cost effective manner, while maintaining high quality, is challenging due to the nature of streaming media: large file sizes, long and continuous bandwidth support, real-time requirements, etc.

These difficulties have slowed down the wide usage of Internet streaming applications. Today, the streaming servers are still very fragile. Compared to milliseconds or seconds for traditional web page delivery, streaming delivery 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, making it extremely fragile.

Thus, today, the majority of Internet media traffic is not through streaming, but downloading. Our most recent study of a snapshot of the Internet traffic in June 2004 [1] shows that currently the overwhelming majority (more than 84%) of media contents are still delivered via downloading from Web servers, in which a substantial percentage of these connections are aborted before completion, resulting in about 20% wasted bandwidth.

Although many studies have been performed and new algorithms and systems have been designed and implemented to improve the Internet streaming, they are rarely used in practice. In addition to the difficulties we have mentioned, the management and investment overhead for streaming software and hardware also prevents streaming services being hosted easily like hosting a Web server.

To deal with these obstacles, in this paper, we have designed and implemented a segment-based proxy, named *SProxy*. Such a system leverages existing Internet infrastructure and enables the content provider to provide streaming service via a common Web server. Upon a client request via the standard RTP/RTSP, *SProxy* communicates with the content-server using HTTP. 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.

Furthermore, *SProxy* uses a segment-aware file I/O system that enables automatic segmentation and intelligent prefetching techniques to guarantee continuous streaming. This allows the *SProxy* to transparently handle the complexity of media formats and to support continuous delivery demands. In addition, it has the following merits. First, a client request is processed and divided into multiple sub-requests. Each sub-request asks for only a small part of the whole media object. The sequence of sub-requests 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. Second, 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.

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

The rest of this paper is organized as follows. We present the design and implementation of the *SProxy* in Section 2. We briefly report evaluation results in Section 3. We make concluding remarks in Section 4.

2. DESIGN AND IMPLEMENTATION OF *SPROXY*

Figure 1 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.

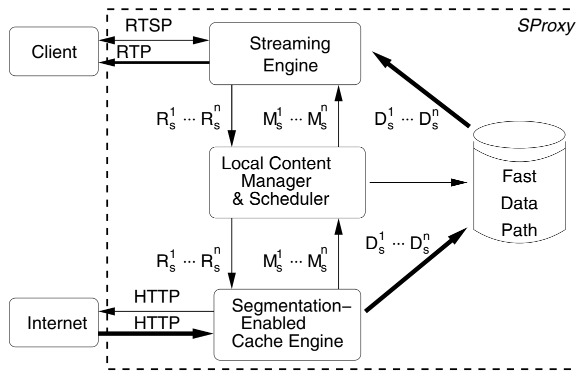

Fig. 1. Internal design of the *SProxy* : A client request is divided into n sub-requests with different ranges, R_s^1 to R_s^n, requesting different content segments, D_s^1 to D_s^n. The Local Content Manager and Scheduler controls when to send the next sub-request. The cache engine returns segment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

2.1. Streaming Engine

The streaming engine is a multi-threaded media server, responsible for providing an interface to the client. Its internal structure is described in detail in [2]. As shown in Figure 1, 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 the LCMS. It uses the meta-data information, $M_s^1 \ldots M_s^n$, returned by the cache engine through the LCMS to access the raw data segments on the disk.

As shown in Figure 1, 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. The LCMS tries to ensure that the next segment is always available in the cache.

2.2. Local Content Manager and Scheduler

The Local Content Manager and Scheduler (LCMS) coordinates the streaming engine and the segmentation-enabled cache engine. It converts the sub-requests, 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 after. More importantly, the 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 it 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. 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.

- *Window*: In this mode, the sub-request 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 sub-request for the next uncached segment is issued after the server has reached the middle of the current one.

- *Active*: Active prefetching is implemented to dynamically decide when to prefetch an uncached segment according to the real-time bandwidths. It is the most precise online prefetching technique according to [3] and is implemented with the aid of Packet CAPture (PCAP) library. The prefetch schedule is based on the media encoding rate and the available bandwidth measured online.

2.3. Segmentation-Enabled Cache Engine

The segmentation-enabled cache engine handles the sub-requests from the LCMS. In case of a cache MISS, the cache engine gets the data for the sub-request 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 the LCMS. The meta data includes the name and the location of the file containing the data for this sub-request on the local disk. In a case of a cache HIT, the cache directly constructs and sends the M_s^n metadata to the LCMS.

Currently, the *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 re-writes the URL internally. For example, a request for:

`http://www.foo.com/bar.mp4`

with

`Range=123-890`

can be rewritten as:

`http://www.foo.com/bar.mp4_123_890`.

This guarantees that different ranges of the same object generate different MD5 keys. This mechanism enables the caching of different segments of a media object.

The re-written 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 re-writing of the URL provides the opportunity to cache the data for different segments of the same object, segment caching is enforced by saving the 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 the multimedia object caching. The *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. The *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 the *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.

2.4. 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, was added to Squid for this purpose:

1. PREFETCH is implemented as a non-blocking 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. The 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(UNLOCK) 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, the LCMS issues a request with a LOCK method. After segment access is complete, the LOCK is released with UNLOCK.

The non-blocking 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.

3. PERFORMANCE EVALUATION

We evaluate the system in both local network (*local* in the figure) and over the Internet (*remote* in the figure). For the Internet experiments, the content server running Apache Web Server (version 2.0.45 with HTTP 1.1) is located in Japan, and our system is located in Palo Alto, CA. The local network environment is set up in a LAN. In both cases, the server runs 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 and the media segment is 100 KB and each experiment was repeated 100 times.

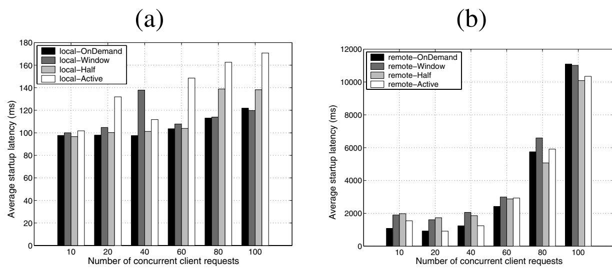

Fig. 2. Client Startup latency for local and remote

Figure 2 (a) shows the startup latency for local accesses, while Figure 2 (b) shows this metric for remote accesses. In the local case, it varies from 96 ms to 169 ms, while for remote accesses, the startup latency is much larger, with a much bigger dynamic range, from 2 s 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. The 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 the *SProxy* can support the delivery of media objects with reasonable startup latency in both intranet and Internet environments.

Figures 3 (a) and (b) show the client perceived jitter in

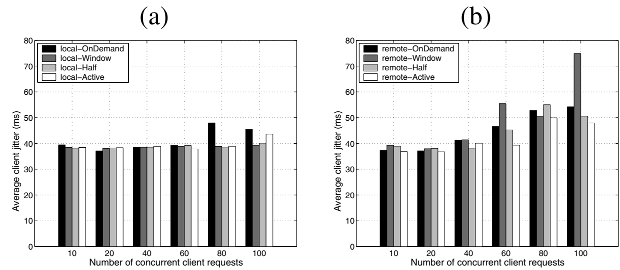

Fig. 3. Client perceived jitter for local and remote

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. CONCLUSION

Recent years have witnessed a lot of streaming service unavailability and a large amount of research work has been conducted to advocate the Internet streaming applications. In this work, we presented our design, implementation, and evaluation of a novel system that enables streaming service from a regular Web server, thus removing the obstacles that has greatly hindered the Internet streaming. Our system has been extensively evaluated and is now deployed in HP Labs.

5. REFERENCES

[1] L. Guo, S. Chen, Z. Xiao, and X. Zhang, "Analysis of multimedia workloads with implications for internet streaming media," in *Proceedings of the 14th International World Wide Web Conference*, Chiba, Japan, May 2005.

[2] S. Roy, J. Ankcorn, and S. Wee, "Architecture of a Modular Streaming Media Server for Content Delivery Networks," in *Proceedings of IEEE International Conference on Multimedia & Expo*, Baltimore, MD, July 2003.

[3] 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 *Proceedings of the 8th International Workshop on Web Content Caching and Distribution*, Hawthorne, NY, September 2003.