ARCHITECTURE OF A MODULAR STREAMING MEDIA SERVER FOR CONTENT DELIVERY NETWORKS

Sumit Roy, John Ankcorn, and Susie Wee

Streaming Media Systems Group
Mobile and Media Systems Laboratory
Hewlett-Packard Laboratories, Palo Alto

ABSTRACT

A Mobile Streaming Media Content Delivery Network can facilitate the access of rich multimedia streams by mobile users on next generation wireless networks. A principal component of this network are the edge servers that cooperate to provide streaming functionalities. This paper describes some of the design requirements for such servers. It then describes the architecture of such a streaming server. Features included are the ability to cache streaming content, a modular design, and an intelligent scheduler for providing better video quality in varying network conditions. Finally, preliminary performance results are shown.

1. INTRODUCTION

The availability of ever increasing bandwidths on the wired and wireless Internet increases user expectations in terms of viewable content. In particular, rich multimedia is poised to be a growing fraction of network traffic. However, streaming media has large storage and bandwidth requirements, and this poses considerable challenges. Delivery of such content to mobile end users is facilitated by constructing a Mobile Streaming Media Content Delivery Network (MSM-CDN) [1]. The basic component of such an MSM-CDN is a streaming edge server or surrogate. A collection of cooperating surrogates forms a *virtual overlay network* on an existing network infrastructure.This overlay network can then be managed to form an efficient content delivery network.

Prior work related work on streaming servers for Video-on-Demand systems [2] has focused on research problems related to optimal resource utilization within a server (eg.

disk access optimization), and bandwidth utilization in the presence of a limited amount of content. However, there are some important differences in the design of a streaming server that forms part of a MSM-CDN. For example, each edge server could cooperate with other servers to form a distributed caching system. The servers may be used to effectively disseminate live media streams. They could also be used by mobile users to upload video streams that then need to be shared with other users.

Another related research area is web caching using proxies. These proxies reduce bandwidth consumption, load at the origin server, as well as latency seen by the client, in addition providing greater fault tolerance in case the origin server is not accessible. However, caching for streaming media sessions has additional challenges. The amount of data is significantly larger than typical web-pages (multiple megabytes instead of multiple kilobytes), and sessions are of much longer duration than typical web-page downloads (of the order of minutes and hours, as opposed to seconds).

This paper describes a streaming server architecture that is designed for a mobile streaming media content distribution network. A system for caching streaming media for stationary clients was described in [3]. In contrast, our architecture is targeted towards mobile clients, and is designed to support a mix of stored streams, live streams, as well as content upload and concurrent sharing.

The rest of the paper is organized as follows. Section 2 elaborates on the essential design requirements. The server architecture is explained in detail in Section 3. Preliminary performance results are shown in Section 4. Section 5 concludes the paper.

2. STREAMING SERVER DESIGN GOALS

A well designed streaming media server for an MSM-CDN would try to meet the following goals:

2.1. Scalable content delivery

Delivering streaming media content to a large number of clients poses considerable challenges. A centralized content server could quickly get overloaded, in terms of I/O requirements. A scalable delivery mechanism requires resources proportional to the number of distinct content items being accessed, rather than the number of clients. Deployment of streaming servers at the edge of the network facilitates this. When multiple clients in the same locality access the same content from a content server, it has to be transferred to the edge server only once, for the first client. After that, the other clients stream the content directly from the edge server. This reduces the load in the core of the network. The various edge servers can also cooperate to further decrease the load on the core network. If the content is not available on a particular edge server, it can query its neighbors, who could be closer than the original content server in a network sense. If the content is found on a neighboring edge server, the client can either be redirected to that server, or the content can be downloaded relatively quickly.

Thus the streaming server should have the ability to proxy streaming content, and to interact with other servers to redistribute and balance load amongst them.

2.2. Responsiveness to changing conditions

Streaming media content requires relatively high bandwidths and stable end-to-end delays, so that the video resolution is acceptable and the frame rate remains constant during playback. Unfortunately, in the current, best-effort Internet, it is difficult to provide such network performance guarantees. This is particularly true when the content servers and clients can have large network distances between them.

The streaming server at the edge of the network can react to local changes in conditions much faster than a content server in the core network. In current generation wireless networks using GPRS, the throughput varies from 16 kbps at the cell center down to 10 kbps at the cell edge, over a distance of 5 km [4]. Thus there is almost a factor of 2 change in available bandwidth due to client mobility alone. A streaming server at the edge could adjust the media stream to utilize current network bandwidth. Moreover, the latency seen by most clients between the media request and the arrival of the first packet is reduced, since the request can be satisfied by the local cache if the content is present. If there is sufficient client side bandwidth, the initial part of the stream can be sent at a higher than required rate, to further reduce the wait time seen by the user due to buffering [5].

Thus the streaming server should be able to react to feedback from the network layer.

2.3. Efficient resource utilization

Streaming media is likely to be accessed by millions of users, which is particularly a problem when streaming live media. If every client sets up an individual session with the original content server, the core network as well as the origin server will be quickly overloaded. The edge servers should therefore have support for splitting a single stream to multiple clients. While IP multicast provides this facility, it requires network operators to enable this feature in their equipment. It is estimated that only 5 – 20 % of the current Internet has multicast support enabled [6]. Application level multicast on the other hand requires no network infrastructure support, since it is done in the overlay network.

Streaming media content also requires a large amount of storage space if the entire clip is to be cached. However, it has been found, that various parts of a clip have different probabilities of being viewed by users [7]. Thus, it is more cost effective to partition the content, and selectively cache the segments, based on their popularity and access pattern. It enables a streaming cache to provide better performance in terms of cache hit ratios and the storage efficiency for a fixed amount of storage [8].

Thus the streaming server should provide support for joining and relaying an on-going streaming session from another streaming server to setup an application level multicast tree. It should also provide support for transparently streaming segmented clips to a media client.

2.4. Live video support

Modern cell-phones come equipped with video cameras. Current GPRS or FOMAnetworks have limited functionality like clip capture and video e-mail. Yet this service provides a rich user experience of casual capture of content. The edge server can enhance this by providing live upload. This lets a user capture content into the infrastructure, avoiding storage limitations of the device. Moreover, the infrastructure can be used to share this content, either concurrently, or delayed. Every user becomes a potential source for publishing personal streaming media content, by making use of the content delivery infrastructure.

Thus the streaming server should provide support for uploading and streaming live content.

3. ARCHITECTURAL DETAILS

A streaming media server in a MSM-CDN can help in scalable content delivery if it has the following features:

3.1. Caching and segmentation support

The streaming server interfaces to a web cache running on the same node, as shown in Figure 1.
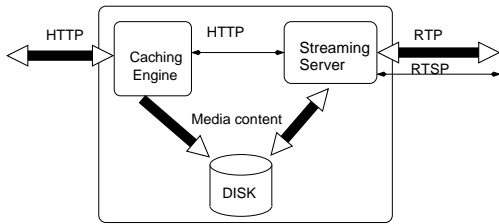
Fig. 1. Caching support

When the server receives a request for some content from a client, typically using the Real Time Streaming Protocol (RTSP), it first queries the local cache for that file. If it is not found locally, the cache can then request an HTTP download of the file, either from the original content server, or from neighboring caches. A useful optimization is that the downloaded file does not have to be transferred between the caching engine and the streaming server. Instead, the cache returns the location in the local file system from where the server can start streaming the content. The streaming server can then send the media data packets, typically using the Real-time Transport Protocol (RTP).

In case the file is segmented, the server generates segment names from the base file name from, and prefetches them before they have to be streamed to a client.

3.2. Modular design

Figure 2 shows that the streaming server consists of multiple modules. There are different types of streams, *File Source*, *Network Source*, *File Sink*, and *Network Sink*. The sources and sinks are linked together through an intelligent *Media Unit Scheduler*.
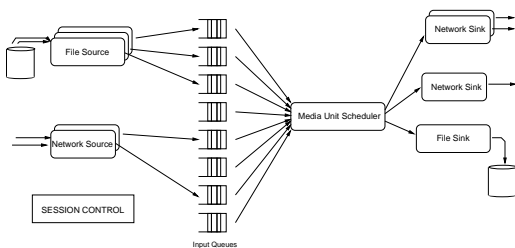

Fig. 2. Modular Design

The *File Source* stream accesses local content from disk. This module can internally schedule disk accesses to provide high throughput for all concurrent streams. In case of temporary input resource starvation, it could statistically drop media packets from different sessions [9]. The *Network Source* has very different operating constraints. It either represents an upstream server, or a client that is trying to record content. In either case, the media unit arrival depends on the upstream scheduler and network conditions.

On the other hand, this type of stream could drop incoming packets that are known to have arrived too late to be useful at any client.

The source streams feed media units into the input queues. These packets are then dispatched to the output streams by the *Media Unit Scheduler*. *Network Sinks* represent typical mobile clients. These clients have packet delivery deadlines so that there is no playback buffer underflow or overflow. A useful feature is to provide feedback about the current network latency and available bandwidth to the scheduler. *File Sinks* are used to save streaming content to local disk. This content would typically be from a live video session. It is to be noted that saving uploaded content does not have any deadlines associated with it.

3.3. Intelligent scheduling

The *Media Unit Scheduler* is used to link together different types of streams. It looks at all the input queues and decides an optimal schedule for transferring media units to the output streams. The objective function should be to minimize the rate-distortion at the client side. If there is sufficient upstream network capacity and disk bandwidth, all packets arrive in the input buffers on time. If there is sufficient downstream network bandwidth, the clients can receive media packets in time for decoding and presentation. In this case the scheduler can use a simple earliest-deadline-first policy. Due to the modular design, the scheduler does not actually need to know the type of input stream.

In case of reduced resource availability, the *Media Unit Scheduler* can take intelligent scheduling decisions. For example, if the outgoing network bandwidth for the server decreases, it can drop dependent video frames and only transmit independent frames. This maintains a better video quality than randomly dropping media packets. The scheduler can also respond to network conditions on a per-client basis, for example by reordering dependent and independent frames in the presence of varying network delays [10].

Since the *Media Unit Scheduler* has a global view of all streaming session handled by the server, it can ensure fairness in terms of victimizing clients when packets have to be dropped, or it can provide better Quality of Service to premium customers.

4. EXPERIMENTAL RESULTS

The prototype server, called the SMSG server, is designed for edge servers. It has not been optimized for pure video-on-demand applications. However, it is useful to compare its performance against a standard streaming server to evaluate whether its feature-rich, modular design introduces any significant overhead.

The server was compared against the *Darwin Streaming*

Server [11]. Each server was run on a *hp workstation x4000*, with dual 2 GHz Intel Xeon processors, 512 MByte RAM running the SuSE 8.0 release of Linux. The client was a *hp Visualize C3600*, with a 554 MHz PA-RISC 8500 processor, 1 GByte RAM, running HP-UX 11.11. The machines were connected via a 100 Mbit switched ethernet network.

In the experiments, multiple clients were started, requesting movie clips from each server at 50 ms intervals. Each clip is 60 s in length. Figure 3 shows the latency seen by each client between sending a PLAY request and receiving the first video packet. It can be seen that the *Darwin* server has a somewhat better performance, presumably due to its state-based design, as opposed to the multithreaded approach of our SMSG server. Figure 4 shows the standard deviation of the packet arrival times compared to an ideal server that could deliver video data at the exact decode time. The streaming server described in this paper does a better job at maintaining accurate delivery deadlines than the *Darwin* server.
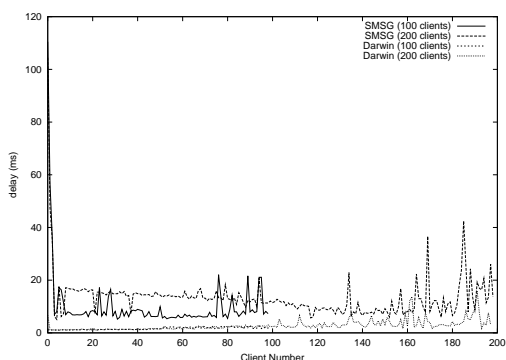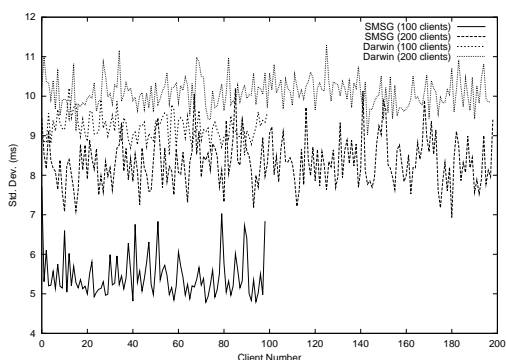

Fig. 3. Response time for each server

Fig. 4. Standard Deviation of Arrival Times

5. CONCLUSION

This paper describes the architecture for a streaming media server in a Mobile Streaming Media Content Delivery Network. The current implementation can cache whole media clips or segmented media. Its modular design allows a seamless mixture of file streaming, client uploads and application level multicast so that multiple clients can see the same media stream. The prototype is being used in a MSM-CDN testbed. Preliminary comparison with a pure video streaming server shows that the prototype has slightly higher response times, but does a better job at providing on-time delivery of media packets. This is particularly important for delay sensitive streaming media applications, like interactive games or video telephony.

6. REFERENCES

[1] S. Wee, J. Apostolopoulos, S. Roy, and W.-T. Tan, "Research and Design of a Mobile Streaming Media Content Delivery Network," in *IEEE ICME*, (Baltimore, MD), August 2003.

[2] D. Sitaram and A. Dan, *Multimedia Servers: Applications, Environments, and Design.* Morgan Kaufman, 2000.

[3] E. Bommaiah, K. Guo, M. Hofmann, and S. Paul, "Design and Implementation of a Caching System for Streaming Media over the Internet," in *IEEE Real Time Technology and Applications Symposium*, 2000.

[4] J. Navarro, J. Martinez, and J. Romero, "Throughput Estimation for EGPRS Services Based on GSM Network Measurements," in *IEEE 55th Vehicular Technology Conference*, vol. 1, pp. 150 – 154, May 2002.

[5] S. Jin, A. Bestavros, and A. Iyengar, "Accelerating Internet Streaming Media Delivery using Network-Aware Partical Caching," in *IEEE International Conference on Distributed Computing Systems*, (Vienna, Austria), July 2002.

[6] Multicast Technologies, "Multicast Status Web Page." http://www.multicasttech.com/status, 2002.

[7] L. Cherkasova and M. Gupta, "Characterizing Locality, Evolution, and Life Span of Accesses in Enterprise Media Server Workload," in *ACM NOSSDAV*, (Miami Beach, FL), May 2002.

[8] 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 Journal on Selected Areas in Communications on Internet Proxy Services*, September 2002.

[9] H. M. Vin, P. Goyal, A. Goyal, and A. Goyal, "A Statistical Admission Control Algorithm for Multimedia Servers," in *ACM Multimedia*, (San Francisco), pp. 33 – 40, October 1994.

[10] S. Wee, W.-T. Tan, J. Apostolopoulos, and M. Etoh, "Optimized Video Streaming for Networks with Varying Delay," in *IEEE ICME*, (Lausanne, Switzerland), August 2002.

[11] Apple, "Darwin Streaming Server 4.1.3." http://developer.apple.com/darwin/projects/streaming/, 2003.