Experiences with MPEG-4 Multimedia Streaming

Hassan Shojania
ATI Technologies, inc.
hshojani@ati.com

Baochun Li
Department of Electrical and Computer Engineering
University of Toronto
bli@eecg.toronto.edu

ABSTRACT

With the advent of next-generation multimedia technologies such as very-low bit rate MPEG-4 codec, multimedia streaming of high-quality video and audio has become a near-term reality. The high compression ratio and error resilience offered by the MPEG-4 standard promise near-term popularity for rich contents **and** exceptional quality to consumers over affordable Internet connections, such as xDSL, cable modem and 3G wireless networks. Audio and video streaming applications are at the center of such scenarios; and Quality-of-Service (QoS) support in such applications is critical to their widespread acceptance.

To the best of our knowledge, there has been no existing open-source MPEG-4 multimedia streaming applications in the academic community, which leads to the lack of research results using MPEG-4 streaming, especially with respect to Quality-of-Service support. In this work, we have implemented an open-source MPEG-4 multimedia streaming testbed in IP-based networks. In this paper, we show our experiences and lessons learned with such a testbed. First, we describe the algorithms and solutions used in our implemention testbed, emphasizing several critical issues. Second, through extensive experiments, we demonstrate measurements of bandwidth requirements and data loss for streaming a set of multimedia samples with different bit rates over UDP, which is ubiquitously available in the TCP/IP protocol stack on all consumer operating systems. Finally, future work for further improvements is also discussed.

1. INTRODUCTION

Streaming media is becoming increasingly prominent over current-generation of broadband IP-based networks. Bandwidth limitations of previous years are not as stringent as before, and content-rich high-quality video and audio are gaining popularity in the form of streamed media. Indeed, high-quality multimedia applications are one of the first choices for utilizing the increasing bandwidth, and one of the propelling forces for researching Quality-of-Service (QoS) issues over the Internet. For example, streaming **delivery** of audio and **video** has become a norm in many web sites for news and educational purposes.

However, since bandwidth availability to the **end user** (the last ***mile*** *problem*) still has severe limitations even with the current high-end technology such as xDSL and cable modem, such streaming are still limited to low-quality video and audio, which is not satisfactory for high-quality home entertainment such video-on-demand. Part of the reasons is that current generation of streaming technology are still using the MPEG-I, MPEG-2, H.261 or a wide range of proprietary codec technologies, which do not offer the compression ratio required to stream high-quality (e.g. near-DVD) multimedia over less than 1000Kbps.

Such advances in bandwidth availability are not limited to wired networks. Third-generation wireless networks are rapidly approaching reality, also providing bandwidth levels similar to that of wired networks. In both cases, end-system resources, especially CPU capabilities, have also advanced in leaps and bounds. In order to provide acceptably high-quality multimedia streaming, the only solution is to trade-off more end-system CPU resources and to stream contents using a different codec that offers higher compression ratios.

The MPEG-4 standard has achieved a unique position in realizing such revolutionary advances with respect to delivering high-quality video and audio to consumers. While most of the current-generation streaming applications are closed and proprietary, the emerging MPEG-4 standard has gained increasing acceptance as the standard for Internet streamed multimedia and has exceptional potential to offer an alternative, open-source streaming solution. Among its highlights and features, it offers highly efficient compression, error resilience, bandwidth scalability ranging from 5Kbits to 20Mbits/second, network and transport protocol independence, as well as content security and object-based interactivity. With respect to streaming, its low-bit-rate audio and video coding capability and its built-in error resilience are especially attractive.

On the other hand, MPEG-4 is considered to be an all-embracing standard, in that different vendors attempt to support a subset using a number of profiles and levels. Some even call it a very ambitious standard without focusing on anything (see [1] for more discussion of this). The industry is split on which MPEG-4 profiles, levels and feature sets need to be supported in applications by servers, client systems and chips. However, with a prototype proof-of-concept testbed for MPEG-4 multimedia streaming, especially if it is also open-source, such deficiencies of the MPEG-4 standard does not prevent the academic community from studying its typical traffic envelopes, loss requirements, bandwidth variations and error resilience levels. To the best of our knowledge, there are no existing work that attempts to study Quality-of-Service issues of streaming MPEG-4 contents over IP-based networks, mostly because of the lack of such an open-source testbed.

In this work, we attempt to break the ice and implement a client/server-based MPEG-4 multimedia streaming testbed over unreliable transport protocols such as UDP. The entire testbed implementation is based on an available open-source MPEG-4 **video** codec [2], with our own open-source streaming extensions. The primary goal is to measure the bandwidth delivered over the network, from the server to the client. We present our choices of algorithms and solutions in such a testbed, and present our

MM'01, Sept. 30-Oct. 5, 2001, Ottawa, Canada.

extensive experimental results based on streaming three real-world high-quality video and audio over the network.

The rest of this paper is organized as follows. In Section II we show the design and architecture of our implementation testbed; in Section III we present extensive experimental results obtained from streaming high-quality media over the network. We conclude the paper in Section IV.

2. Algorithms In Testbed Implementation

In this section, we present the algorithms and solutions we have adopted in our client/server-based MPEG-4 multimedia streaming testbed. In addition, we summarize our lessons learned and experiences from implementing such a testbed. In details, we first introduce the basic structure of an MPEG-4 player based on an open-source MPEG-4 codec; followed by discussions related to our own streaming extensions based on UDP, which is ubiquitously available in the TCP/IP protocol stack on all consumer operating systems.

2.1 The OpenDivX MPEG-4 Codec

Our work is based on an open-source MPEG-4 player just released in January 2001, referred to as *The Playa*. The *Playa* is implemented on the Windows platform. It is currently capable of playing local MPEG-4-coded files through the support of an open-source MPEG-4 codec that conforms to the MPEG-4 standard, referred to as *OpenDivX* [2]. In addition to MPEG-4 video, *The Playa* is able to play audio in various formats using Windows-based *DirectSound* interfaces (e.g. PCM, MP3). The playback process can be divided into the following two phases:

Phase 1: Preparation

All basic information about video/audio properties, such as video bitmap information (size and color depth), frame rate and codec types is retrieved in this stage. The list of video/audio headers in the MPEG-4 file is then parsed and data about position in the file and length of each video frame (or audio chunk) is collected for easy access in later phases.

Based on the information gathered at previous phase, proper interfaces to audio and video codecs are created and initialized. Video playback surfaces are then created.

Phase 2: Playback Loop

Outstanding video frame is read from the MPEG-4 file, decompressed and then rendered. This process is accomplished in a new thread, leaving the main thread responding to user interface events. The thread waits if video playback is ahead of its time, or drops some frames if it's backlogged. Audio decompression and playback are done in a separate thread. Note that "dropping" here only means "not rendering". Each frame must be decompressed till the codec states are updated for decoding subsequent frames.

2.2 UDP-based MPEG-4 Streaming

For the purpose of extending the MPEG-4 file player and accommodate multimedia streaming over the network, we have created two variants of the *Playa*, one functioning as a server and the other as a client. The server opens a file and waits for a connection from the client. After receiving the connection, it streams video and audio using UDP packets to the client with the same rate as if it would do the playing Itself. We have attempted to minimize our interaction with the *Playa* algorithms to make future integrations to its newer versions easier. As a preliminary proof-of-concept testbed, our current implementation does have some limitations. For example, the server supports only one client; and no random access to the content is available, i.e. the clip must be played sequentially from the beginning.

We have chosen straightforward UDP as our underlying protocol, since it is ubiquitously present in any TCP/IP stack implemented in all consumer operating systems. Since using UDP implies data loss or out of order delivery, buffer management at the client side becomes the most challenging part of our work. We present additional details as follows.

2.2.1 Server-side Implementation

After a file is selected, regular extraction of header and index information is initiated. This is immediately followed by the creation of two sockets, one connection oriented (TCP) and the other one connectionless (UDP). The server then waits for client's connection to deliver the header and general information about the clip over the connection-oriented socket. Regular playback loop starts thereafter, but whenever a video frame or audio chunk is read from file, the data is initially copied to our send buffers for transferring at correct intervals. We utilize two 64KB buffer as send buffers, one for audio and the other for video. Video frames or audio chunks are appended into their related send buffers and whenever either buffer is full (i.e. cannot accept the next frame or chunk), it is sent to the client with a blocking send on the UDP socket. A partially filled buffer might be sent earlier if the interval between two successive send operations is more than a threshold. This is necessary to generate data continuously when video or audio bit rate is low (i.e. black frames at start of a film or a good quality stereo MP3 audio with only 16KBytes/s). This threshold is currently set at 0.5 second.

Each video frame or audio chunk appended to the send buffers is preceded with a header including its timestamp, size and miscellaneous flags for extra processing needs.

2.2.2 Client-side Implementation

Client implementation is more complicated since it needs careful and efficient buffer management to make sure no data is lost because of delayed processing in the receiving phase. This might happen because of the unreliable nature of UDP delivery. Besides, if outdated data is not invalidated from buffers on time, we might run out of the available buffer and lose new data.

In our current testbed, 150 buffers are utilized, each with the size of maximum frame size in the clip'. Since the *Playa* reads audio data in chunks of 2048 bytes from the file, proper number of 2048 bytes buffers is allocated to match the number of video buffers (for the same duration of ahead buffering).

Similarly, the client creates a connection-oriented and a connection-less socket. It receives the tile header and general information on the TCP socket. Using this information, it initializes the data structures of original code to emulate retrieval from a file. It then launches the receiving thread, which issues

---

This maximum frame size is transferred from server to client as part of general information sent on the initiating TCP connection

successive receive commands on the UDP socket. Each packet received is processed and its corresponding audio or video buffers are updated. Buffers form ordered double-linked lists (for both audio and video) to ease traversing of buffers, when addmg new frames at receiving time and retrieving existing frames at playback time.

The playback is not started till 4/5 of video buffers arc filled. It results in around 4-5 seconds (for 23.97 to 29.97 FPS samples we had) of ahead buffering before starting playback. Video and audio threads of the original player run their regular codec algorithms. Only when a read of video frame (audio chunk) is requested, the video (audio) list is searched for that specific frame (chunk). If the frame (chunk) is found in the buffers, it'll be decompressed and rendered.

After a video frame (audio chunk) read is processed (successful or not), all other outstanding video frames (audio chunks) with equal or lesser timestamp are invalidated in the buffer lists and returned to the available buffers pool. To filter out-of-date incoming frames (chunks), we also check the timestamps of the arriving data with the timestamp of last played data and ignore them if they are delayed.

3. Analysis and results

We have performed extensive experiments with respect to *data throughput* and loss for streaming three real-world near-DVD quality multimedia clips, "The Matrix", "Jurassic Park" and "Frankenstein", including both video and audio. As expected, the data loss was acceptably low on a 10Mbps LAN. The CPU capabilities on the client side have a significant effect on the data loss rate. For high bit-rate samples on a Pentium III 500MHz CPU, the decoding process increases CPU utilization up to 100 percent for most parts of the clip, resulting in failures of retrieving data from the transport layer on time, which leads to considerable data loss. We have since upgraded the client's CPU to 733MHz, which leads to a significant reduction in data loss for high bit-rate samples.

Table I shows the results of running UDP delivery of our multimedia clips to the client. Some of the losses are attributed to high bit-rate segments of the clip or the start of playback (i.e. losing some packets when decompression of first frames is started on client, around second 5 of clip). However, we are unable to argue the same for all data losses. Additional analysis of client logs and bit rate distribution of the clip is necessary. Some of these losses happen when CPU usage was quite low (e.g. 45%), so it might be that our algorithm is not issuing receive commands on time, or due to temporary back-offs on the local area network.

TABLE I
STREAMING DELIVERY OF SAMPLE CLIPS

Sample & resolution[2]	Compression rate[3]	Number of video frame (audio chunk) loss/total [4]	Avg. video bit rate/ Max rate[5] (Kb/s)	Avg. audio bit rate	Length
Matrix-4 720*480	5000 Kb/s	14 / 37872 (0 / 12343)	2248 / 6699	I26	26 min.
Matrix-3 720*480	6000 Kb/s	16 / 26907 (0 / 8768)	2283 / 7570	126	19 min.
Jurassic-3 720*480	900 Kb/s	13 / 34882 (0 /11361)	915 / 2034	126	24min.
Jurassic-3 720*480	2000 Kb/s	11 / 34864 (0 / 11361)	1997 / 3214	126	24 min.
Jurassic-3 720*480	5000 Kb/s	IO / 10966 (0 / 3574)	4038 / 7309	126	8min.
Jurassic-3 720*480	6000 Kb/s	29 / 34845 (0 / 11361)	4156 / 9882	126	24 min.
Frankens-tein-1 720*480	900 Kb/s	8129312 (0 / 9554)	980 / 2680	126	20 min.
Frankens-tein- 1 720*480	5000 Kb/s	217219 0 / 2352	2897 / 727 1	125	5 min.
Frankens-tein- 1 320*240	900 Kb/s	112 / 131427 9 / 42873	727 /1576	125	91 min.

[1] The test environment:

- Server: PIII-550MHz, *Windows 2000*
- Client: PIII-733MHz, *Windows 2000*.
- *OpenDivX* version 4.0 Alfa 48 for MPEG-4 video codec
- For all samples, audio was coded using Fraunhofer MPEG Layer-3 codec at 128Kbit/s.
- 10Mbps LAN, not highly loaded (after-hours condition)

[2] All samples were 23.97 FPS.

[3] Compression rate selected on *MPEG-4* codec.

[4] Note that effects of one frame loss are sometimes noticeable for next few seconds till the next key frame updates the codec states.

[5] Maximum bit rate is measured at the server side (from input file).

Experiments under more complicated network conditions such as a more congested LAN or over high-speed Internet can be among our next tasks. Better audio and video synchronization is necessary when high bit rate samples are played on a low-end CPU (e.g. PIII 500MHz). In such cases, the original synchronization method implemented in the *Playa* does not perform well when video decompression lags far behind audio. A more aggressive approach may be required in these cases.

4. Concluding Remarks

In this paper, we have presented an open-source MPEG-4 network streaming testbed for synchronized audio and video, based on existing open-source MPEG-4 codecs. It is implemented as a proof-of-concept testbed for the purpose of supporting subsequent academic research with respect to the analysis of real-world MPEG-4 streaming over IP-based networks. The fact that our code will be open source is a significant support to the academic community to migrate from MPEG-I or MPEG-2 centered research to the MPEG-4 standard, particularly with respect to Quality-of-Service issues.

5. References

[I] J. Yoshida, "MPEG-4: key enabler or 'science fiction'?," EE TIMES, Issue 1154, pp. 1-16, Feb. 19, 2001.

[2] Project Mayo website, http://www.projectmayo.com.