A Server-centric Streaming Model[1]

Jin Hwan Jeong, Chuck Yoo

Department of Computer Science and Engineering, Korea University

jhjeong@os.korea.ac.kr, hxy@joy.korea.ac.kr

Abstract

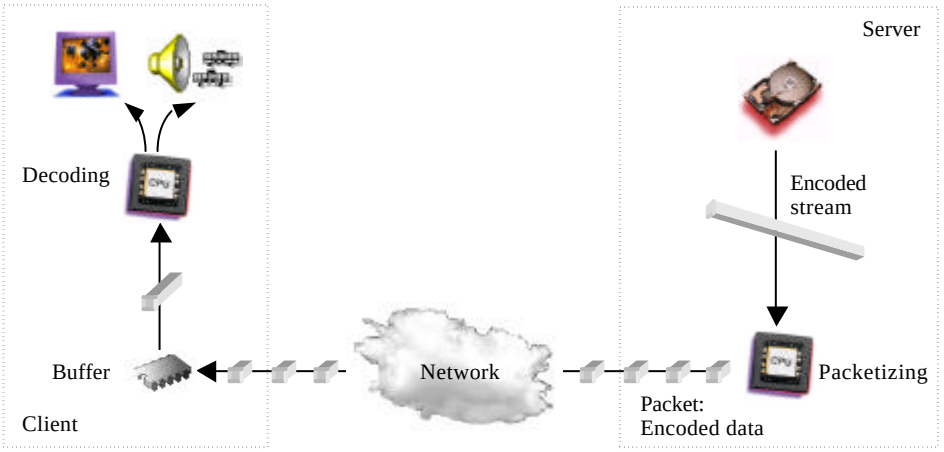
The current streaming technology is based on a model that a server sends encoded multimedia files and that a client does decoding and rendering in real time. In this model, a client must have a powerful hardware and must have specific decoders to handle various media file formats (e.g. MPEG, H-263, etc.). This paper starts with a different assumption: network bandwidth is available so that it is no longer a bottleneck. Based on this assumption, we present a new streaming model where the server participates in decoding – we call it intermediate decoding. We explore the different levels of intermediate decoding, which lead to the tradeoff between the processing power required at client and bandwidth consumed. The new model is applied to MPEG-1, and the measurement results are presented.

1. Introduction

Advances in multimedia technology make it possible to store moving images in digital form. However, even with sophisticated encoding techniques, the storage requirement is too huge. For example, a movie of 15 minutes easily occupies about 150M bytes of disk space. It is difficult to carry such a large file, and it is obviously burdensome to store it in local disks of PC even though disk capacity improves constantly. Media streaming is getting popular and widely used because it removes the storage requirement. As depicted in Fig. 1, with media streaming, a server stores the media files and transmits encoded "streams" to clients that want to view. Then a client decodes the encoded stream and synchronizes the media types (e.g. audio and video) in order to play. The advantage of this streaming model is that a client does not need to store the received media files. It just buffers the media data in memory for a period of time, long enough to avoid network jitter and delay, and throw it away after playing.

The assumption behind the streaming model is that the speed of Internet is slow. The bandwidth that a stream can use is limited to the slowest physical line between the server and client. It can be a part of congested backbone link or a dial-up link. By sending encoded streams, the bandwidth requirement is minimized, which makes the streaming model applicable to the Internet. However, the implication of this model is that the server merely pumps up the compressed files and the responsibility of handling the compressed stream is on clients. We can say that the streaming model is client-centric.

1. This research was supported in part by Sun Microsystems Laboratories.



[Figure 1. Current streaming model]

This paper starts with a different assumption: network bandwidth is available so that it is no longer a bottleneck in streaming. As justification of the assumption, nowadays we see dramatic improvement in high-speed network technology. The gigabit and fast Ethernet become a commodity, and the new Internet backbone (such as Internet 2) is promised to be much faster than today's speed. Digital Subscriber Line (DSL) is expected to solve the last mile problem. This trend motivates us to explore a new streaming model, which is the goal of our paper.

Specifically, Fig. 2 is the envisioned model where the server does "intermediate" decoding of compressed streams and transmits the intermediate data. The client decodes the intermediate data and plays it. The difference from the current streaming model is that the role of client is much simplified. We call this server-centric model. By applying the new model to the streaming of MPEG-1 data, we analyze the trade-off between the bandwidth and processing power in the server-centric model.

This paper is organized as follows. Section 2 describes the limitations of existing streaming software. The server-centric model is explained in details in Section 3. The intermediate data for MPEG-1 is defined, and the synchronization in the server-centric model is presented. Section 4 presents the implementation and the results of experiments, which is briefly given due to the lack of space in this extended abstract.

2. Limitations of existing streaming software

The most famous examples of the current streaming software are the Real Player of Real Networks, the Stream Works of Xing Technology, and the Media Player of Microsoft. Although they support different encoding formats, the core algorithm is the same because they follow the client-centric model: the server sends encoded data in their own format that is highly compressed, and the client decodes the received data with a decoder that knows how to uncompress the format.

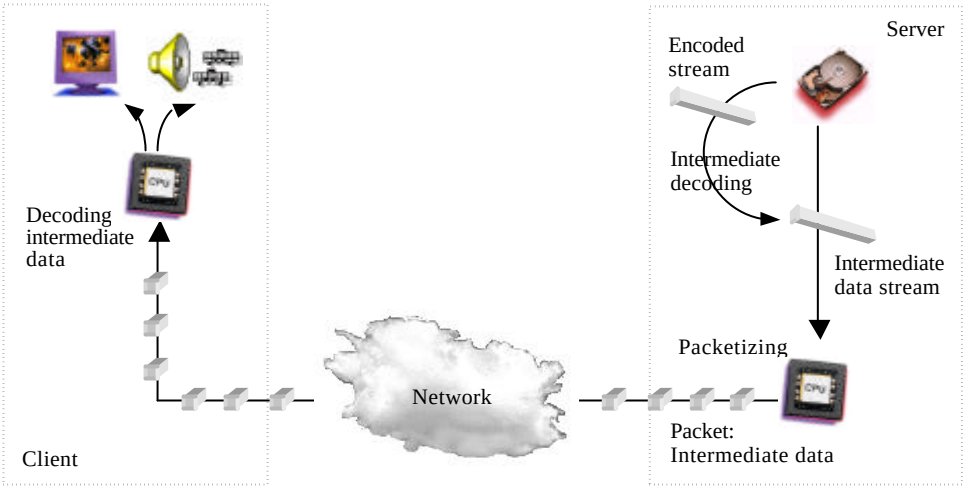
However, the client-centric model has a few disadvantages. First, it is difficult to simplify the client. It means that the client should have hardware powerful enough to process the streams in real time because

the client has to do decoding and synchronizing and rendering all together. For software decoding, CPU should be fast enough. Even with the latest CPU, however, it is quite easy to see the whole CPU occupied with software-decoding of sophisticated compression algorithms such as MPEG-2. Therefore, most PDA or thin client cannot play the compressed streams due to the lack of CPU power. Second, extensibility - if a client wants to handle both of MPEG and H.263 streams, it should have two separate decoders because the encoding is different so that specific decoders are unavoidable in order to handle different media formats. Third, they are not able to take advantage of emerging high bandwidth networks because the streams are typically encoded at low bit rate (56Kbps), which is targeted for dial-up connection. When a server does streaming with 56K-encoded files, the quality of rendered images is the same regardless of the network bandwidth of the connection. To resolve this problem, a server has several versions of file with different encoding rates or has a layered media file. However, these approaches make it difficult to maintain the server and have difficulties in handling layered media files properly with the dynamically-changing network traffic.

3. Server-centric streaming model

3.1 Intermediate decoding

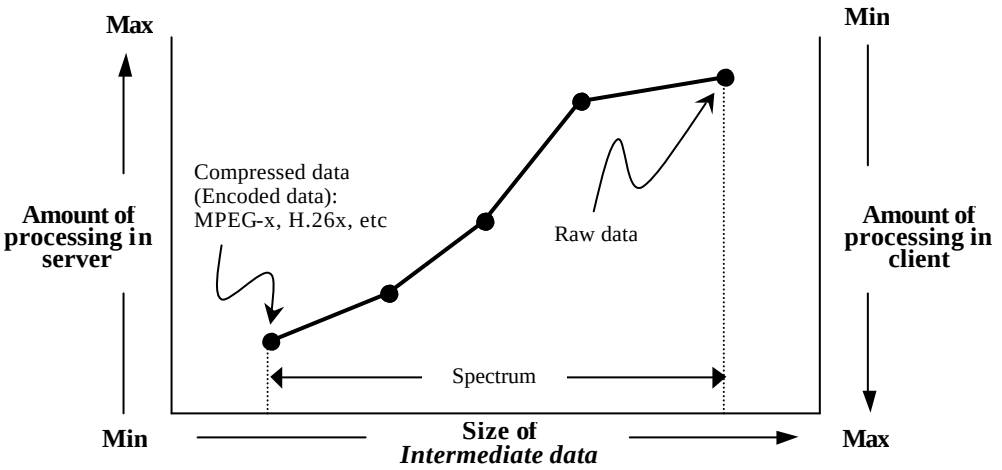
Our goal is to overcome the disadvantages of the client-centric streaming model. First, we want to have a streaming model that can be applied to "thin" and portable devices as well as "fat" devices. We will see more and more of various types of thin and portable devices, and thus the streaming need for such devices is quite clear. The client-centric streaming model has the orientation for fat devices. Second, even for fat devices, it is more desirable to have a generic decoder than having a decoder for each media data format. There are already many different media data formats and there will be new ones to come. Therefore, we want to have a generic decoder to cover the existing media formats and future ones if possible.



[Figure 2. Newstreaming model]

The key of the server-centric streaming model is intermediate decoding and intermediate data. The idea of intermediate decoding is to divide the decoding steps into two phases. The server decodes the first phase and the client does the second phase. The intermediate data is the output generated from the first decoding phase in server. In the client-centric model, a client goes through all the decoding steps. But in the server-centric model, the server is involved in decoding. The server does the decoding steps to generate intermediate data and sends the intermediate data. Then the client produces the raw data from the intermediate data. To achieve our goal stated above, decoding of the intermediate data at the client should be simple enough for thin devices, and the intermediate data is common for as many compression algorithms as possible.

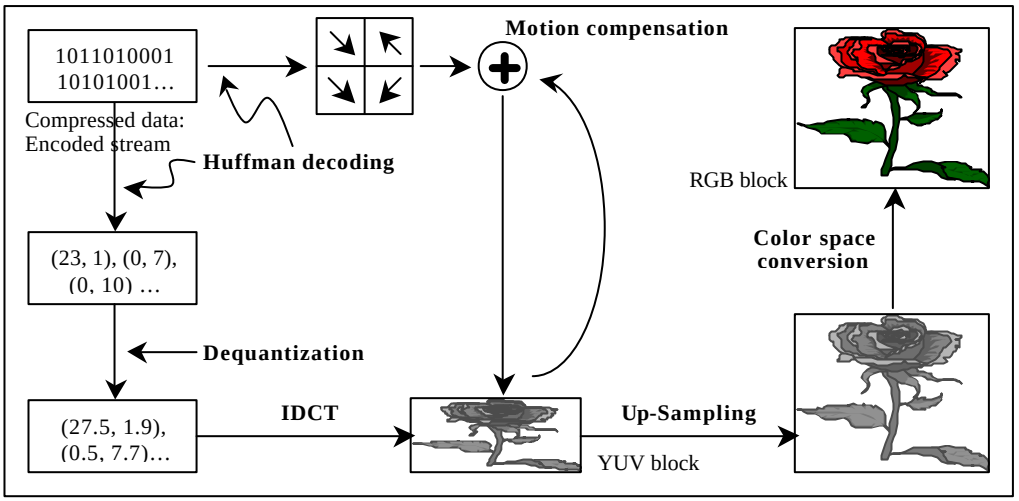
Now question is: how to define intermediate data. Our basic idea is to define the intermediate data in the decoding steps of block-based compressions (where JPEG, MPEG-x, H.261, H.263, etc. belong). The intermediate data can be anywhere between compressed data and RGB raw data. There is a spectrum from compressed data to raw data where intermediate data can be defined as shown in Fig. 3. We believe that nobody has explored the spectrum other than the compressed data for streaming. We analyze the effects of the different intermediate data (points in Fig. 3) in the spectrum.



[Figure 3. Spectrum of intermediate data for streaming]

As the intermediate data is closer to RGB raw data, the server does more decoding and the intermediate data size gets larger, which lead to higher bandwidth requirement. But the client does less decoding. So there is a trade-off depending on the level of intermediate decoding - how much decoding to be done at the client and the bandwidth to be consumed. To understand the trade-off, we are measuring the bandwidth and the CPU time in each decoding step of MPEG-1.

MPEG-1 decoding steps can be divided into Huffman decoding, IDCT, Dequantization, Motion compensation, Up-sampling, and Color space conversion. The data after Huffman decoding is done can be an example of intermediate data. Or the output of IDCT or IDCT and dequantization steps can be an intermediate data respectively. The measurements results will be included in the final paper



[Figure 4. Decoding steps of MPEG-1]

3.2 Synchronization

A difficult problem we have to address in the server-centric model is the synchronization of media types. In the traditional model, a server sends encoded media files in the form of so-called system stream where various media types are multiplexed, client can do decode and play them with the synchronization hint in the system stream (e.g. Present Time Stamp in MPEG-1). In contrast, in the new streaming model, since the server does decoding, it sends an intermediate data generated from each media type (video, audio, etc.) whether they are multiplexed or not. So clients have no idea about how to synchronize various intermediate data (e.g. video intermediate data, audio intermediate data, etc). Possible solutions are described in the final paper.

4. Implementation and Performance

We implemented an MPEG-1 player based on the new streaming model and applied it to Sun Ray 1 enterprise appliance. The reason why we chose Sun Ray 1 is that Sun Ray 1 is one of thin client machines that are low power computers. Sun Ray 1 includes four major hardware components: 100MHz microSPARC-IIep processor that is equivalent to 486-class CPU, 10/100Mbps ethernet device, frame buffer controller with ATI Rage 128 chip, and 8MB of memory. For effective transmission, the server uses UDP protocol.

The result of experiments shows that our player can play 352 X 280 video clip (display dimension is 1056 X 840) at full frame rate on Sun Ray 1. This result emphasizes the fact that our streaming model is applicable to thin devices like Sun Ray 1 because it requires the minimum system resources especially CPU time and memory usage. The final paper will include more of the measured performance data.