Wireless Networks 11, 265–274, 2005

A Video Streaming System for Mobile Phones: Practice and Experience

HOJUNG CHA* and JONGMIN LEE
Department of Computer Science, Yonsei University, Seodaemun-gu, Shinchon-dong 134, Seoul 120-749, Korea

JONGHO NANG and SUNG-YONG PARK
Department of Computer Science, Sogang University, Mapo-gu, Shinsu-dong 1, 121-742, Korea

JIN-HWAN JEONG, CHUCK YOO and JIN-YOUNG CHOI
Department of Computer Science, Korea University, Sungbuk-gu, Anam-dong 5, Seoul 136-701, Korea

Abstract. This paper presents a case of video streaming system for mobile phone which has actually been implemented and deployed for commercial services in CDMA2000 1X cellular phone networks. As the computing environment and the network connection of cellular phones are significantly different from the wired desktop environment, the traditional desktop streaming method is not applicable. Therefore, a new architecture is required to suit the successfully streaming in the mobile phone environment. We have developed a very lightweight video player for use in mobile phone and the related authoring tool for the player. The streaming server has carefully been designed to provide high efficiency, reliability and scalability. Based on a specifically-designed suite of streaming protocol, the server employs an adaptive rate control mechanism which transmits the media packets appropriately into the network according to the change in network bandwidth.

Keywords: wireless video streaming, video codec, mobile phone, streaming server, authoring tools

1. Introduction

As high-speed wireless network technologies such as GPRS and CDMA2000 1X become a reality, cellular phone is expanding its role to be a computing device, not just a voice communication device, and people are exploring to run various applications on cell phone [5]. However, the computing environment of cellular phones is so different from the desktop processors that direct porting of the desktop applications to cellular phone often finishes as a failure. The difference exists in all aspects—CPU speed, memory size, display dimension and so on. And importantly, porting itself is technically a big problem because few debugging tools are available; in other words, the computing environment is similar to that of early computing systems a couple decades ago. Even though cell phone has many limitations, it is believed as a key device in pervasive computing because the number of cell phones being sold is much more than the desktop system and the growing number of people is using cell phones for data communication to get connected to the Internet. In order to make cell phones even more "pervasive", running Internet-based applications are of prime importance. Among the Internet-based applications, video streaming is particularly hard for two reasons: frequent packet losses and lack of processing power of handsets. Unlike the fixed-line networks, the wireless networks have frequent packet losses as well as high RTT (round-trip time) [7]. The packet loss is quite persistent regardless of time, and the rate is relatively high (5–10%) by noise. This causes numerous jitters and image distortion while video clips are streamed and played, which makes existing streaming methods not applicable. Furthermore, existing streaming methods use video standards such as H.26x or MPEG, but the handsets do not have processing power to decode and play compressed bitstreams properly.

The major challenges for mobile streaming service are to support a variety of access networks and terminals, and provide the services in an uniform way. The Third Generation Partnership Project (3GPP) is currently addressing mobile streaming standardization which includes both protocol and codec [3]. The standardization is, however, somewhat heavy to apply the standard technologies to the current mobile infrastructure. The industry trend is rather towards mobile streaming services based on proprietary protocols and codecs. There are a few research and development effort for mobile streaming systems. For instance, AT&T [2] and Fabri and Kondoz [4] proposed streaming video systems for the EGPRS wireless networks. Based on H.263 video streams, AT&T developed a mobile streaming system and proposed a rate selection algorithm which selects an appropriate streaming rate depending on the channel quality. Fabri et al. developed a general streaming system and several techniques for guarantying QoS. These works are, however, based on a wide range of performance assumptions and verified via simulations.

The main contribution of this paper is to present comprehensively the design and implementation details of a system architecture that makes video streaming possible on the state-of-the-art cell phones and the mobile networks. The architecture has been actually deployed and presently services a large number (over a million) of cell phones. Figure 1 illustrates

* Corresponding author.
E-mail: hjcha@cs.yonsei.ac.kr

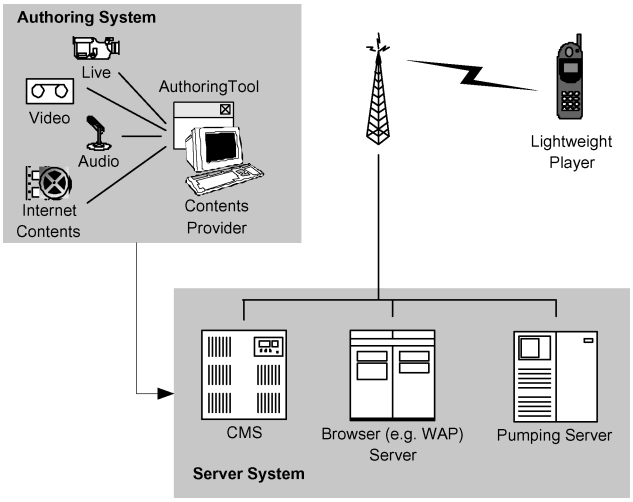

Figure 1. System overview.

the overall architecture for the mobile video service system developed in our work. The mobile phone is equipped with a browser and a lightweight video player. With this phone, a subscriber connects to the server that contains video contents. The server of the browser located in the server system provides the content list in the form of browser menu. A video clip is selected from the menu, and the clip is streamed or downloaded from the streaming server. The content authoring system converts video in existing format into our proprietary LVF (Lightweight Video Format) files suitable for the video player, and the file is uploaded into the server. The architecture consists of three key components: the lightweight video player running in a cell phone, the authoring system for the player, and the server system for pumping bitstreams to the cell phone. This paper discusses each component in details.

The paper is organized as follows. Section 2 discusses the lightweight video player. Sections 3 and 4 describe the server system and the authoring system respectively. The paper concludes in Section 5.

2. Lightweight video player for mobile phone

This section discusses the lightweight video player we have developed for use in mobile phone. The dithering issue and the decoding process on mobile phone are also discussed.

2.1. LVF video codec

A popular processing core of cell phones is ARM7 [1]. It is a RISC processor with UV pipeline which can execute one instruction per clock cycle. Although a handset is in idle state, it always runs some of the call processing tasks such as searching a home agent, communication with a base station and so on. Therefore, the available CPU cycles for actual applications are much less than what the processor provides. The available CPU processing power is typically about one fourth of the original power and this is equivalent to only a few MIPS. When a handset starts to run data communication protocol such as TCP/IP, the available CPU cycles would obviously become more scarce. Also as the amount of memory in cell phone is very limited, it is difficult to buffer the streamed data in an efficient manner. Buffering is a natural assumption in most streaming algorithms found in the literature, but a typical cell phone is not equipped with an enough memory to store an ample amount of data. So the techniques such as the bi-directional prediction are not useful.

One possible approach is to use an extra hardware to speed up the video processing on cell phones. This approach is, however, considered inappropriate in our research as the cell phones are battery-operated and the use of extra hardware component would make the battery life short. Consequently, what we have been looking for is a software-based codec solution which minimizes both the processing power and the memory consumption on cell phones. We have analyzed several video compression algorithms available in terms of the processing requirements and the compression ratio [8]. Among considered are the entropy coding techniques—such as the run length coding(RLE), Huffman coding, Arithmetic coding—and the transform coding techniques—such as the discrete cosine transform (DCT) and the wavelet transform. RLE is a very simple encoding algorithm with low compression ratio. Huffman and the arithmetic coding, on the other hand, result in higher compression ratio, but they require more processor cycles. The transform coding technique such as DCT improves the compression with quantization [9]. The well-known MPEG compression algorithm, for instance, follow the sequence of 'complex' compression through the sub-sampling, transform coding, quantization, and the modified Huffman coding phases. Recently there has been active research on the wavelet transform coding. The computation complexity of the wavelet transform varies according to its basis function [6]. After carefully analyzing relationship between the compression ratio and the computational complexity of various coding techniques, we have developed a wavelet-based video codec, LVF(Lightweight Video Format), which reduces the computational complexity of video coding/decoding so as to run it on existing cell phones.

Basically LVF is one of the block-based hybrid video codecs such as MPEG-x and H.26x. However, as these codecs require substantial decoding complexity, we have modified the key sub-coding algorithms used in typical block-based hybrid video codecs: the transform coding and the entropy coding. Instead of using the CPU intensive DCT which is hardly acceptable for handset processor, LVF uses a wavelet function for the transform coding. It is well understood that a wavelet function runs very fast while yielding low bit rate and it is adequately used for even real-time video coding on low-end hardware such as the handset processors. As for the entropy coding, LVF uses a modified version of RLE. Although the Huffman decoding is popular for the entropy coding, its CPU requirement is non trivial; for instance, every access for the codeword table on ARM7TDMI with no CPU cache generates a memory reference. RLE is another alternative. RLE has,

however, a shortcoming of compression ratio—especially in the case the run length becomes 1 frequently. That is, if there are many coefficients with run length of 1, the data size can be larger than the original data size. To overcome this drawback, the normalization step is inserted between the quantization phase and the entropy coding phase. LVF normalizes the coefficients generated by quantization phase and then runs the RLE coding. This way, all coefficients have, after normalization, common points such as even number or multiple of 3. Consequently the RLE with the normalization step add-on outperforms the original scheme as the encoding complexity reduces.

Table 1 and 2 shows the performance of LVF compared to H.263. Both LVF and H.263 are implemented on a Pentium III 500 MHz processor. Three video clips of the canonical form are used for the experiment, "Miss America(MA)", "Foreman(FM)" and "Carphone (CP)", and the original clips are encoded with four different compression rates. Two metrics are used for the performance measurements: the CPU processing time and PSNR for the image quality. The figures in the table explain that the image quality of LVF is generally worse than that of H.263, but the processing overhead is greatly reduced with LVF. At the compression rate of 55 kbps, for instance, the PSNR value of LVF is approximately 4–5 dB lower than that of H.263. However, by sacrificing the image quality of 5 dB, LVF has reduced the processing time about 25 times (164 ms/4070 ms) over H.263. The 5 dB difference of PSNR could be meaningful, but this degree of quality degradation becomes in reality negligible when we consider the fact that the images are rendered on resolution devices such as the handset LCDs (typically 112×96). Overall, the experimental results state that the wavelet-based LVF delivers a dramatic processing time improvement with only a little quality degradation and hence it is lightweight enough to be used for handset applications.

Table 1
PSNR comparison (unit:dB).

	Compressed bitrate (bps)	55,563	29,489	21,375	17,915
MA	H.263	48.1	45.3	44.7	44.4
	LVF	43.0	38.0	35.3	33.5
FM	H.263	44.8	42.2	40.8	39.3
	LVF	40.9	34.4	30.8	28.5
CP	H.263	47.4	44.6	42.7	41.5
	LVF	41.4	35.3	31.9	29.8

Table 2
Decoding time comparison (unit: milliseconds).

	Compressed bitrate (bps)	55,563	29,489	21,375	17,915
MA	H.263	4,070	2,906	2,608	2,542
	LVF	164	153	146	135
FM	H.263	14,668	11,652	9,908	8,872
	LVF	711	652	626	586
CP	H.263	11,028	10,227	8,561	7,780
	LVF	590	565	555	507

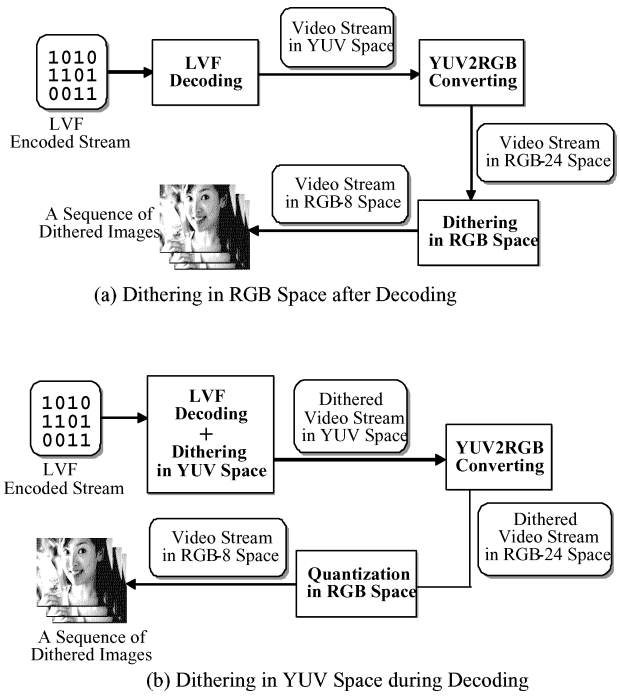

Figure 2. Dithering on mobile phones.

2.2. Dithering on mobile phone

A common type of the off-the-shelf cell phones has a display device with different number of colors; e.g., gray scale LCDs, 256-color or 65536-color LCDs. The video decoder running on cell phones should therefore perform dithering for a better image quality. Dithering is a process of converting 24-bits device-independent images into appropriate image formats suitable for LCD display. The dithering algorithm is usually applied to a RGB image obtained from the up-sampled YUV pixels as shown in figure 2(a). As dithering algorithms usually work on the whole pixels on every frame, the computation requirement is not trivial. Hence, it is necessary for a video decoder to support the dithering mechanism at the decoding level rather than at the rendering level as shown in figure 2(b). Here, "decoding level" means that the output of the inverse function of the wavelet transform becomes a dithered YUV image. This approach can reduce the memory operations and the arithmetic computation associated with RGB conversion and up-sampling which are critical to performance and power consumption. The core of the dithering in decoding level is to compute the dithering and the inverse function of the wavelet transform simultaneously. This way, the dithering overhead is completely absorbed into the computation of the inverse function. The decoder developed in our work is based on an ordered dithering algorithm which only requires pixel coordinates and pixel values. Figure 3 shows an example of dithered images on mobile phone with different color depths.

2.3. Decoder architecture

The operating systems running on cell phones should be minimal and provide an optimal way of handling preset task

Figure 3. Example of dithered images on mobile phones with different color depths.

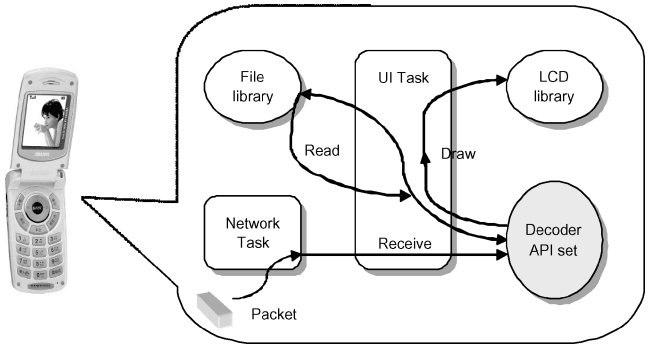

Figure 4. Decoder architecture for cell phones.

priorities based on an efficient task-based scheduling policy. In order not to intervene the underlying operating system's scheduling principles, we have implemented our codec as an API library and attached it to the UI (User Interface) task. The UI task accesses the fundamental system services and as an LVF player in our implementation. It encompasses the decoder API set, in addition to the management facilities for stored data, network stream and LCD rendering. Figure 4 illustrates the decoder architecture for cell phones. The overall decoding process is as follows: The UI task initializes the running environment of the decoder API library—for example; creating timers for decoding and for rendering, file descriptors, UDP network connection and so on. When the decoding timer is expired, the UI task invokes the decoder API which decodes incoming stream and subsequently stores reconstructed images in UI frame buffer with PTS (Presentation Time Stamp). In the case of the rendering timer expired, the UI task checks a frame buffer and renders the reconstructed images on appropriate PTS. For streaming, the UI task receives network packets, stores it to the decoder buffer, and follows the same processes as the local decoding.

3. Wireless streaming server

This section describes the server issues for the wireless streaming system we have developed. The server architecture is described first and the rate control policy used in our implementation is then presented.

3.1. Structure of streaming server

The streaming server is targeted for multi-CPU platforms which can support thousands of concurrent media streams. The server is designed for high efficiency, reliability and scalability. Figure 5 illustrates the server structure. The server consists of admission controller, resource manager, load balancer and task pool. The number of tasks in a task pool is equivalent to the number of processors in the server hardware. A task allocated to a processor is in charge of actual media streaming. The task consists of disk manager, network manager, buffer manager, message handler and task manager. A user's service request is processed as follows. The listener detects a new service request. The admission of this request is decided by the admission controller. The decision is based on the availability of system resources. That is, the admission controller admits a request only when the extra system resources required by the new request are available in the system. This is because the streaming server has to guarantee QoS of the existing sessions currently under services. The system resources are maintained by the resource manager and typically include cpu usage, disk bandwidth, network bandwidth and memory size. The resource manager has to maintain an up-to-date usage information for various system resources. Once admitted, the request updates the resource usage and a processor to execute the request is selected by load balancer. By monitoring the system loads, the load balancer distributes the requests to processors appropriately to maintain the load level of each processor equal. The task manager in a task is responsible for scheduling the stream request to meet its timing constraints.

The stream service model used in our work is a server-push model. The server should, therefore, handle clients with different bitrate characteristics in a consistent manner to guarantee their deadlines. Upon receiving a request from client, the server reads a fixed-size media block from disk into buffer, analyzes the stream and transfers it to network according to its media rate.

The video content stored in the storage consists of a sequence of media packets. Each packet has a timestamp which denotes the latest time it should be shipped into network. The local timestamp of media packet is added to the system time, which is managed by the stream scheduler (i.e., task manager), and then the media packet is inserted into the time-lined job queue where the requests are already sorted according to their final timestamps. The scheduler examines the queue periodically and transmits all the media packets whose timestamps are within the current scheduling period. Here, the scheduling period is called T_{schedule}. Figure 6 illustrates the concept of stream scheduling in our work.

Although the server is designed to use the system resources effectively, the client's service is not guaranteed when the

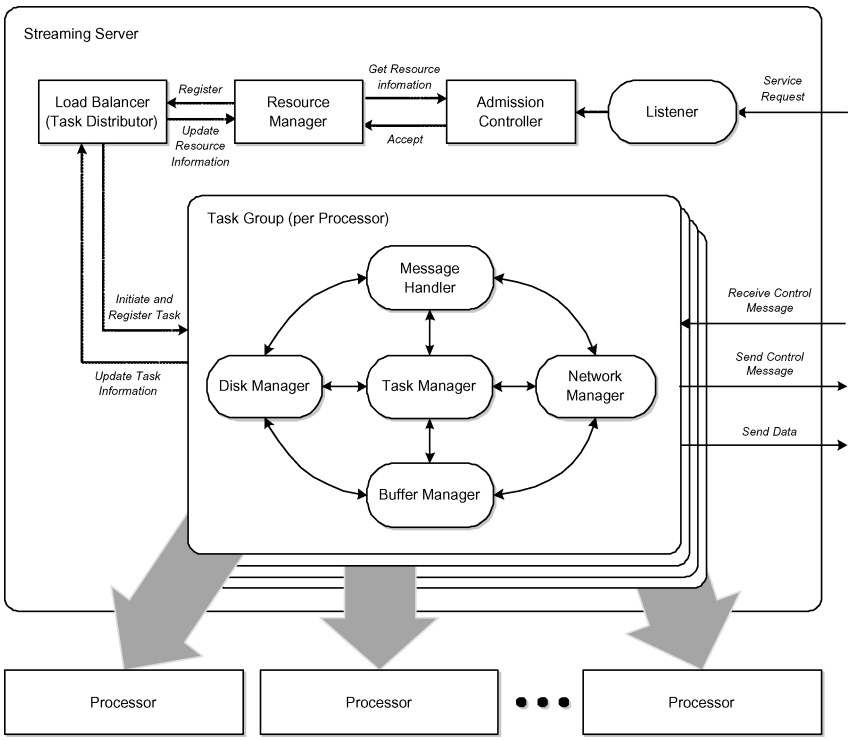

Figure 5. Server structure.

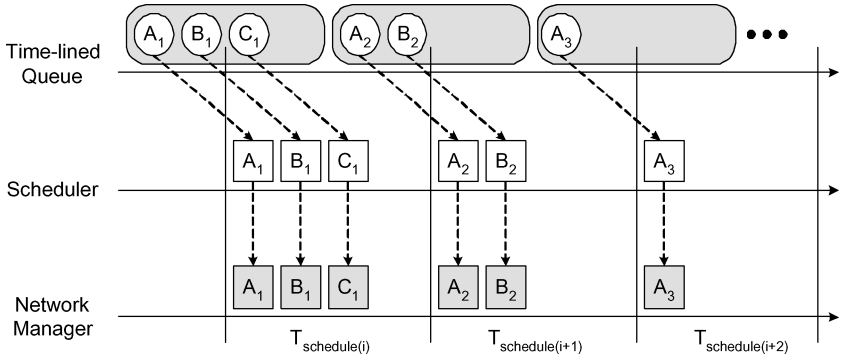

Figure 6. Stream scheduling.

system capacity is exceeded by allowing excessive number of clients. A streaming system should be provided with an adequate admission control mechanism, which decides the admission of a new request, based on the current resource availability of the server. The decision should, of course, not violate the QoS requirements of clients already in service. The admission control used in our work is based on the current system load as well as the resource requirements of a new client. The resource requirements of a new stream is known *a priori*. The system loads such as processor usage and disk access bandwidth may, however, vary dynamically depending on the client behaviors and the underlying operating systems. The server periodically checks the resource usage of system components and updates them in the resource table. This resource information is used as a basis for admission control. Multiple criteria are used for the resource availability: processor usage, memory capacity, disk and network bandwidths.

3.2. Control policy for wireless streaming

The handset such as the mobile phone has many practical limitations for use in media streaming: especially, the CPU processing power and the memory size. For streaming services, the handset has to decode video stream while receiving the network packets. It is therefore necessary to minimize the processing cost for handling the network packets to provide a better video quality. Optimizing the network-related software for the handset is one approach. The careful management of the streaming protocols and policy at the streaming server can also reduce the overhead on the handling of network packets in handset. The actual media streaming or download is done with a specifically designed suite of communication protocols. A TCP-based control protocol is used between the server and mobile phone to exchange control messages. The data transport protocol is based on UDP for streaming and TCP for

download, respectively. The streaming uses UDP as its base protocol since the timely delivery of media packet is important, due to the reduced protocol overhead of UDP, in streaming applications. For download, however, a TCP-based protocol is used since the download operation should provide a lossless media delivery.

The media streaming server implemented in our work uses an adaptive rate control mechanism with which the media packets are pushed into the network with varying intervals—adaptive to the changes in current data transfer size and the available buffer size at the handset. The rate control mechanism consists of three key parts. First, the original media stored in the form of multiplexed video and audio streams is splitted into separate media types and then pushed into the network. This is to reduce the demultiplexing overhead in the handset. Second, the splitted audio and video data is packed into a network packet whose size is determined by the size of P-MTU (path maximum transmission unit). This procedure at the server helps the handset to reduce the computational resource in assembling small fragmented network packets into data packet for decoding. Third, the network packet is pushed into the network according to the relative deadline, not the absolute deadline, of media time which reflects the current status of the mobile handset. Multimedia data streams often have variable bit rates. Pushing media packets into the network simply following its time stamp could result a network congestion and consequently causes packet losses or media delays. In mobile environments, it is particularly important to provide a streaming bandwidth as constantly as possible since the changes in streaming bandwidth may result an unpredictable delays in allocating and freeing the wireless channels. To maintain a constant streaming bandwidth, our adaptive mechanism calculates the media transmit interval appropriately for each network packet, based on the constantly updated time stamps which reflect the status of the handset. The adaptive rate control mechanism used in our work is outlined as follows:

```
Adaptive_rate_control_for_streaming_service(){
   Calculate_network_packet_size();
   Pack_media_data_into_network_packet();
   Get_handset_state();
   Calculate_new_timestamp_for_network_packet();
   Add_network_packet_to_streaming_scheduler();
```

In the case of download services, the server should employ a media transport mechanism which makes use of the available network bandwidth efficiently for the maximum throughput. The underlying transport protocol used for download applications is typically the connection-oriented and reliable TCP. In the wireless and mobile environments, however, the packet loss frequently happens due to the transmission error as well as the network congestion. In particular, the packet loss caused by the transmission error mistakenly activates the congestion control mechanism of TCP and consequently the actual throughput tends to decrease. In order to cope with the communication characteristics of the mobile environment, we have implemented an application-level adaptive rate control mechanism for media downloads. It is similar to the congestion control mechanism used in TCP in a way that the media transmit rate varies depending on the changes of the underlying network condition. The rate control mechanism starts pushing the network packet with the base rate of R_B. If the network packets are successfully transmitted, the push rate is increased by R_I. If the packet transfer fails, however, the push rate is decreased by R_D. Figure 7 illustrates the adaptive control mechanism used in the download application. The streaming server dynamically controls stream the push rate. The initial push rate R_B is derived from the initial push period P_B. Similarly, R_I and R_D are obtained from P_B and P_D, respectively. R_I and R_D are not constant values, but depend on the changes in P_B, P_I and P_D. The figure shows the relationship between the push rate and the push period.

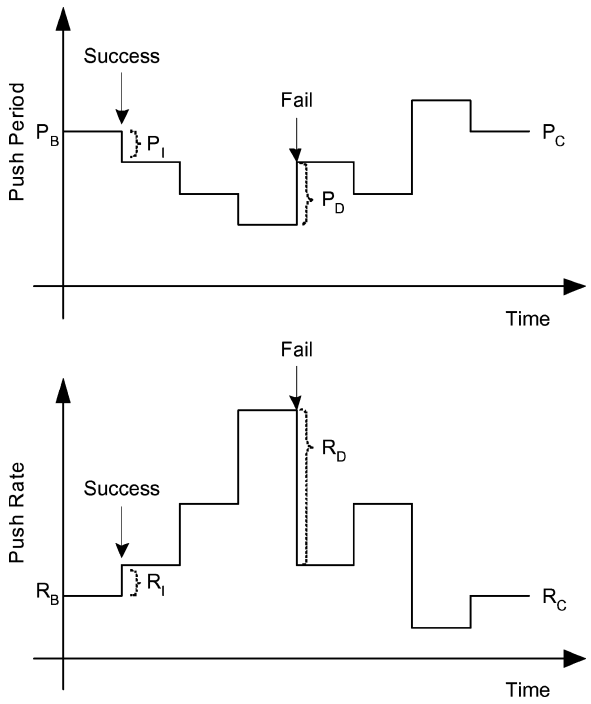

Figure 7. Adaptive download rate control.

The actual values of R_B, R_I and R_D for the underlying service network can be experimentally measured and determined. In our experiment, the streaming server is physically located in the private network of a commercial service provider. By hosting the server in a relatively optimal environment, the packet loss or the network delay due to the congestion in the wired portion of the networks can be minimized. The Samsung CDMA2000 mobile phone (X270) is used as a mobile terminal for media download. In order to analyze the performance effect of various rate control mechanisms on the application-level download throughput, we have experimented two kinds of adaptive mechanisms (*RateControl1* and *RateControl2*) and compared their performance with the mechanism with no rate control (*NoControl*). The network packet size used for the experiment is 1460 bytes which is the maximum data packet size (i.e., excluding protocol headers) that

is not fragmented on Ethernet. The value R_B is set 23 kbps and it is the average throughput which has been experimentally obtained in our environment. The initial push period P_B is, therefore, $\frac{1460\times 8}{23} = 500$ milliseconds. Equations (1) and (2) respectively show the values of P_B, P_I and P_D which are used for *RateControl1* and *RateControl2* in the experiments. Here, P_C and R_c stand for the current push period and the current push rate. Note that *RateControl2* has smaller P_D, i.e. a smaller R_D, than *RateControl1*. In other words, *RateControl2* decreases the push rate more slowly than *RateControl1* in the case of transmission failure.

$$P_B = 500,\ P_I = 50, \quad P_D = \frac{P_C}{2} \quad \text{(msec)} \qquad (1)$$

$$P_B = 500,\ P_I = 50, \quad P_D = \frac{P_C}{3} \quad \text{(msec)} \qquad (2)$$

Figure 8 shows the throughput comparison for the three mechanisms. For each case, 10 experiments have been conducted. The result shows that the throughput of *RateControl2* is approximately 50 and 40% better than those of *NoControl* and *RateControl1*, respectively. This means that the adaptive rate control mechanism is indeed effective and increases the application-level throughput. The experiments also reveal that the parameters should be carefully selected for optimal performance.

4. Contents generation systems

This section presents the basic transcoding mechanism and various authoring tools used in our implementation.

4.1. Basic transcoding mechanism

The main task of authoring tool is to transcode the video clips in digital forms into streams for our lightweight video player. However, since the digital video clips usually compressed

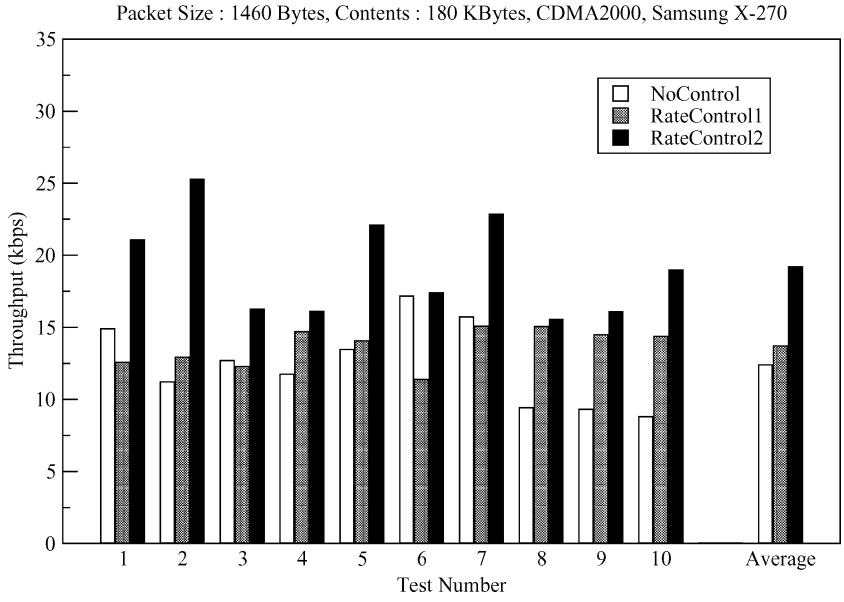

Figure 8. Throughput comparison.

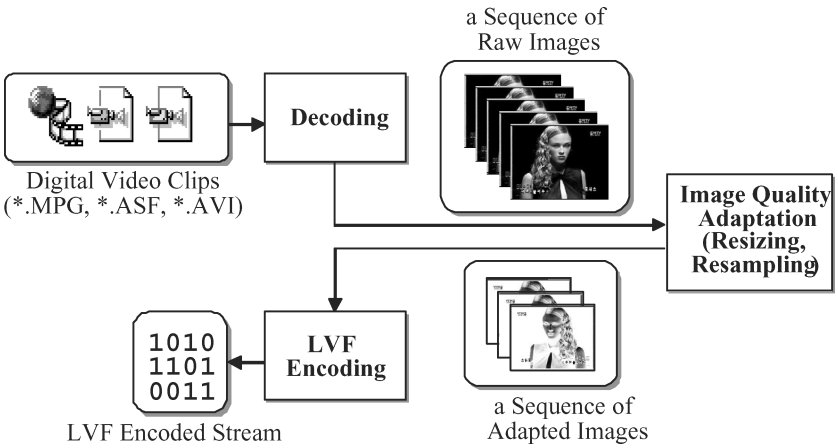

Figure 9. Translation process.

in various formats such as MPEG-1/2/4, AVI, ASF, and WMV, it would be very hard to develop separate transcoders that work in compressed domain directly. Therefore, we have decided to use an approach to decode the source video clips into a raw format first, and then re-encode it to our format. In this transcoding process, however, since the source video clips are usually generated for the desktop computers with a high quality display (for example, 1280 × 1024 pixels, true color (2^{24}-bit color), 30 frames/second), their resolution, color depth, and frame rate should be adjusted accordingly in order to be displayed on mobile phones that have a small display size (for example, 112 × 96 pixels), a limited color depth (for example, 4-level gray or 256 color), and a limited computing power (for example, 3–4 frames/seconds). Figure 9 shows the main processing steps to transcode the source video clips to our file format. Let us explain each step in more details.

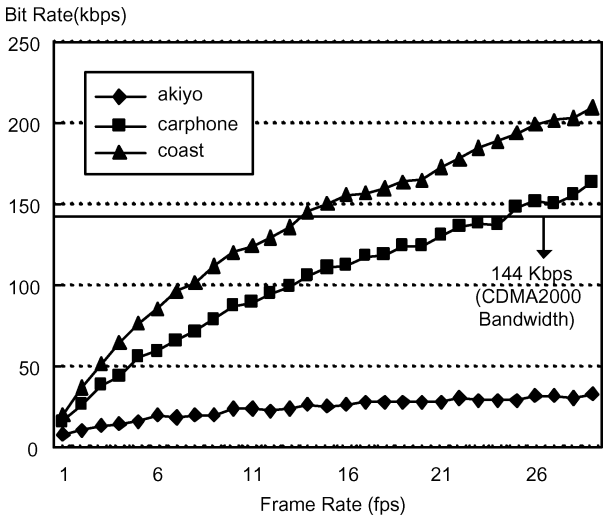

Figure 10. Example of bitrates of LVF-encoded video stream.

Decoding. In order to convert the compressed video clips into a sequence of raw images, the video clips should be decoded first. This task could be easily accomplished with the Microsoft DirectX technology on Windows environment. A filter graph is dynamically configured with respect to the compression format (or type) of the source video clip, and it decodes and produces a sequence of raw images. This approach helps us to transcode any video clips that could be decoded and displayed with Microsoft media player on Windows into our file format.

Image Quality Adaptation. After the video stream is converted into a sequence of raw images, it should be adapted to be displayed on mobile phone. The adaptation process includes the resizing step which changes the resolution of the images, and the re-sampling step which sub-samples the image frames. The decoder in the mobile phone also performs the dithering task which reduces the color depth of image according to the target display device. Let us explain each step in more details.

The resizing step sub-samples the pixels in the video frames to reduce the resolution of the target video stream (usually, 112 × 96 pixels) in RGB space. If the resolution of the source video stream is not a multiple of sixteen, some pixels are truncated for the efficient wavelet-based encoding. The re-sampling step reduces the frame rate of the source video stream with respect to the wireless network and mobile phone constraints. Figure 10 shows an example of the bit rates of three video streams (112 × 96 pixels, true colors) that have different motion activities and encoded with our LVF encoder, in which "akiyo" video stream has a lot of motion activities. As shown in this figure, the bit rate of the video stream is proportional to the frame rate and the amount of motion activities. If we consider CDMA2000 wireless network whose bandwidth is 144 kbps, the video streams with more than 10 fps could be serviced although there are a lot motion activities in the video stream. However, as explained in the previous section, the bottleneck of the wireless video service is the client computing capability, not the wireless network bandwidth. It forces us to reduce the frame rate of video stream to less than 10 frames per second. Currently, the authoring tool statically generates several versions of adapted video streams with different frame rates. One of them is selected with respect to the client's QoS (such as client types), and pumped to client by the streaming server.

Since the LCD display device of the mobile phones is usually darker than CRT or TFT terminals for the desktop computers, an image processing technique that adjusts the brightness and contrast of the image is also required to improve the visual quality on mobile phone. Some other image quality enhancement techniques such as histogram-equalization are also helpful to enhance the image quality of the video clip on mobile phone.

LVF Encoding. A sequence of dithered and sub-sampled images is encoded by our LVF video encoder, while the audio part of video clip is decoded and converted into PCM format (8KHz sampling, 16-bit LVF, mono sound) first with Microsoft DirectX technology, and then re-encoded with EVRC format by audio encoder. These two encoded streams are multiplexed with appropriate time stamps by our encoder. The encoded stream is saved as a file or delivered to the mobile phone directly for the streaming services.

4.2. Authoring system

Figure 11 shows the graphical user interface of our authoring tool. It has many functional characteristics. The authoring tool supports various transcoding options with which many QoS parameters such as frame rate, resolution, color depth, brightness/contrast/gamma values, and audio volume are adjusted. This adjustment could be applied to whole video clip or only to selected sequence. The tool also allows the author to select

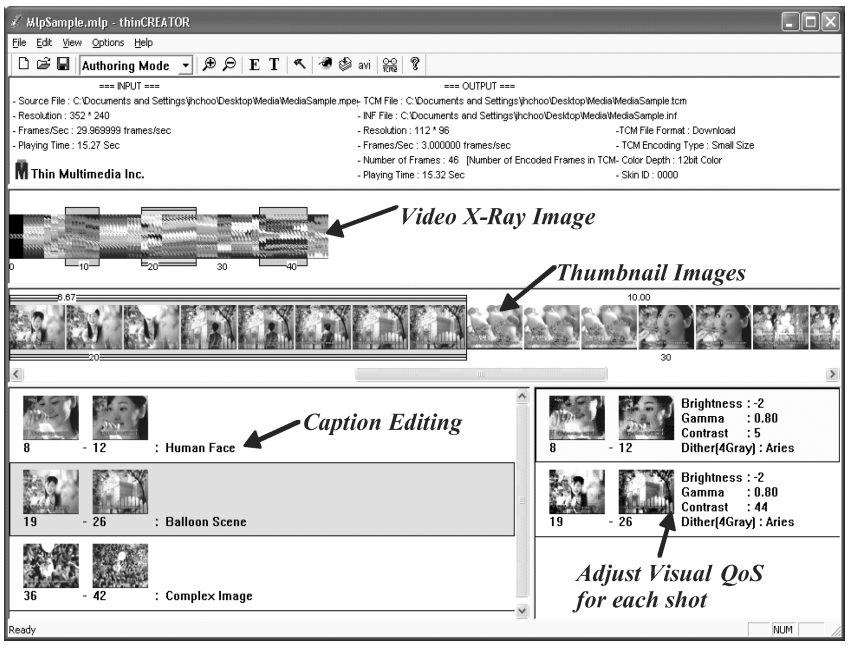

Figure 11. Graphic user interface of the authoring tool.

the sub-sequence of video frame and attach a caption that delivers additional information on video clips. In order to easily browse the video clip, the authoring tool provides two video browsing views: a sequence of thumbnail images and video X-ray image which is a collection of the pixels in the first (or last) line (or column) of each image. The tool also provides a mechanism to detect the shot boundary automatically and it helps encode only the key frames of the video clip. The generated contents could be previewed on PC with the same quality as on mobile phone.

5. Conclusion

The mobile streaming system presented in this paper has actually been implemented and commercially deployed nationalwide, by one of the major wireless carriers, in the Republic of Korea. The number of LVF-enabled cell phones deployed in the country are currently more than a million and we expect that the number will grow sharply. As there are several types of cell phones supporting different color space, the contents providers make their video clips (combined audio and caption) in gray (4-level) or color (256, 4096, 65536 colors) formats and upload them into the server. People can receive and play video clips stored in the server via their LVF-enbled cell phones. A typical video clip is about 150 KBytes long (approximately 30 seconds of playback duration) and it normally takes less than 30 seconds to receive the whole clip in CDMA2000 backbone. The server has been implemented to support 1,000 concurrent streams and thoroughly tested to meet this requirement. As of this writing, the daily traffic is in the order of a few tens of thousands connections and the peak usage is about 100 connections per minute. In this commercial service environment, the daily server statistics persistently shows that the cases of successful content streaming reaches approximately 95% of the total streaming accesses.

Acknowledgments

This work was supported by the Basic Research Program of the Korea Science and Engineering Foundation (grant No. R01-2002-000-00141-0), and the ITRC Program (MMRC) of IITA, Korea.

References

[1] ARM Ltd., http://www.arm.com.
[2] K. Chawla, Z. Jiang, X. Qiu and A. Reibman, Transmission of streaming video over an EGPRS wireless network, in: *International Conference on Multimedia and Expo (ICME) 2000* (New York, July 2000) pp. 275–278.
[3] I. Elsen, F. Hartung, Uwe Horn, Markus Kampmann and Liliane Peters, Streaming technology in 3G mobile communication systems, IEEE Computer 34(9) (2001) 46–52.
[4] S.N. Fabri, and A.M. Kondoz, Provision of streaming media services over mobile networks, in: *2nd International Conference on 3G Mobile Communication Technologies* (London, UK, March 2001) pp. 104–108.
[5] L. Garber, Will 3G really be the next big wireless technology?, IEEE Computer 35(1) (2002) 26–32.
[6] A. Graphs, An Introduction to Wavelets, IEEE Computational Sciences and Engineering 2(2) (1995) 50–61.
[7] L. Hanzo, Interactive cellular and cordless video telephony: state-of-the-art system design principles and expected performance, Proceedings of IEEE 88 (2000) 1388–1413.
[8] J.H. Jeong and Chuck Yoo, A server-centric streaming model, in: *Proceedings of the 10th International Workshop on Network and Operating System Support for Digital Audio and Video* (Chapel Hill, June 2000).
[9] D. Legall, MPEG—A video compression standard for multimedia applications, Communications of the ACM 34(4) (1991) 46–58.

Hojung Cha is currently a professor in computer science at Yonsei University, Seoul, Korea. His research interests include multimedia computing system, multimedia communication networks, wireless and mobile communication systems and embedded system software. He received his B.S. and M.S. in computer engineering from Seoul National University, Korea, in 1985 and 1987, respectively. He received his Ph.D. in computer science from the University of Manchester, England, in 1991.

E-mail: hjcha@cs.yonsei.ac.kr

Jongmin Lee is a Ph.D. candidiate in computer science at Yonsei University, Seoul, Korea. His research interests include wireless multimedia system, QoS architecture, multimedia communication networks. He received his B.S. and M.S. in computer science from Kwangwoon University in 1999 and 2001, respectively.

Jongho Nang is a professor in the Department of Computer Science at Sogang University. He received his B.S. degree from Sogang University, Korea, in 1986 and M.S. and Ph.D. degree from KAIST, in 1988 and in 1992, respectively. His research interests are in the field of multimedia systems, digital video library, and Internet technologies. He is a member of KISS, ACM, and IEEE.

Sung-Yong Park is an associate professor in the Department of Computer Science at Sogang University, Seoul, Korea. He received his B.S. degree in computer science from Sogang University, and both the M.S. and Ph.D. degrees in computer science from Syracuse University. From 1987 to 1992, he worked for LG Electronics, Korea, as a research engineer. From 1998 to 1999, he was a research scientist at Telcordia Technologies (formerly Bellcore) where he developed network management software for optical switches. His research interests include high performance distributed computing and systems, operating systems, and multimedia.

Jin-Hwan Jeong received the B.S. and M.S. degrees in computer science from Korea University, Seoul, Korea, in 1997, and 1999, respectively. He is currently in Ph.D. course at Korea University. His research interests include video processing for thin devices, multimedia streaming and operating systems.

Chuck Yoo received the B.S. degree in electronics engineering from Seoul National University, Seoul, Korea and the M.S. and Ph.D. in computer science in University of Michigan. He worked as a researcher in Sun Microsystems Lab. from 1990 to 1995. He joined the Computer Science and Enginnering Department, Korea University, Seoul, Korea in 1995, where he is currently a professor. His research interests include high performance network, multimedia streaming, and operating systems.

Jin-Young Choi received the B.S. degree from Seoul National University, Seoul, Korea, in 1982, the M.S. degree from Drexel University in 1986, and the Ph.D. degree from University of Pennsylvania, in 1993. He is currently a professor of Computer Science and Engineering Department, Korea University, Seoul, Korea. His current research interests are in real-time computing, formal methods, programming languages, process algebras, security, software engineering, and protocol engineering.