# MPEG Video Streaming with VCR Functionality

Chia-Wen Lin*, Jeongnam Youn**, Jian Zhou†, Ming-Ting Sun†, and Iraj Sodagar‡
*Department of Computer Science and Information Engineering
National Chung Cheng University, R.O.C.
cwlin@cs.ccu.edu.tw
**Equator, USA
jnyoun@equator.com
†Department of Electrical Engineering
University of Washington, USA
{zhouj, sun}@ee.washington.edu
‡PacketVideo, USA
iraj@packetvideo.com

## Abstract

*With the proliferation of online multimedia content, the popularity of multimedia streaming technology, and the establishment of MPEG video coding standards, it is important to investigate how to efficiently implement an MPEG video streaming system. Digital video cassette recording (VCR) functionality enables quick and user friendly browsing of multimedia content, thus is highly desirable in streaming video applications. The implementation of full VCR functionality, however, presents several technical challenges which have not yet been well resolved. In this paper, we investigate the impacts of the VCR functionality on the video decoder complexity and the network traffic. We propose a least-cost scheme for the efficient implementation of MPEG streaming video system to provide full VCR functionality over a network with minimum requirements on the network bandwidth and the decoder complexity. We also discuss our implementation of an IP-based MPEG-4 video streaming platform which provides full VCR functions.*

## 1. Introduction

Multimedia applications have entered an exciting era that will enormously impact our daily life. Today's multimedia technology allows network service providers to offer versatile services such as home shopping, games, video surveillance, and movie on demand [1-2]. In these applications, video streaming technology plays a critical role in the media delivery. Realizing that video streaming has so many applications and so great commercial potential, many companies, organizations, and universities are developing new products [3-7], standards, and technologies [8] in this area.

A video streaming system should be capable of delivering concurrent video streams to a large number of users. The realization of such a system presents several challenges, such as the huge storage capacity and throughput in the video server and the high bandwidth in networks to deliver the video streams. With the rapid progress in processing hardware, software, storage devices, and communication networks, these problems are being solved and video streaming applications are becoming increasingly popular.

In addition to supporting a large number of users, with the proliferation of online multimedia content, it is also highly desirable that multimedia streaming systems support effective and quick browsing. A key technique that enables quick and user friendly browsing of multimedia content is to provide full VCR functionality [9]. The set of effective VCR functionality includes forward, backward, step-forward, step-backward, fast-forward, fast-backward, random access, pause, and stop (and return to the beginning). This set of VCR functionality allows the users to have complete controls over the session presentation and is also useful for other applications such as video editing.

With the establishment of MPEG video coding standards [10-12], it is expected that many video sequences for streaming applications will be encoded in MPEG formats. However, the implementation of the full VCR



0-7695-0933-9/00 $10.00 © 2000 IEEE

functionality with the MPEG coded video is not a trivial task. MPEG video compression is based on motion compensated predictive coding with an I-B-P-frame structure. The I-B-P-frame structure allows a straightforward realization of the forward-play, but it imposes several constraints on other trick modes such as random access, backward play, fast-forward play, and fast-backward play. Straightforward implementation of these trick modes requires much higher network bandwidth and decoder complexity compared to those required for the regular forward-play function.

With the I-B-P-frame structure, to decode a P-frame, the previously encoded I/P-frames need to be decoded first. To decode a B-frame, both the I/P-frames before and after this B-frame need to be first decoded. To implement a backward-play function, a straightforward implementation at the decoder is to decode the whole group of picture (GOP), store all the decoded frames in a large buffer and play the decoded frames backward. However this will require a huge buffer in the decoder to store the decoded frames which is not desirable. Another possibility is to decode the GOP up to the current frame to be displayed, and then go back to decode the GOP again up to the next frame to be displayed. This does not require the huge buffer but will require the client machine to operate in an extremely high speed which is also not desirable. The extra memory and computational costs soon become unaffordable when the GOP size is large.

Besides the problem with backward-play, fast-forward/backward and random-access also present difficulties. When a P/B-frame is requested, all the related previous P/I-frames need to be sent over the network and decoded by the decoder. This requires the network to send all the related frames besides the requested frame at a much higher rate which can be many times of that required by the normal forward-play. When many clients request the trick-modes, it may result in much higher network traffics compared to the normal forward-play situation. It also requires high computational complexity in the client decoder to decode all these extra frames.

Some recent works have addressed the implementation of VCR functions for MPEG compressed video in streaming video applications [1,13-15]. Chen *et al.* [1] described a method of transforming an MPEG I-P-B compressed bit-stream into a local I-B form by performing a P-to-I frame conversion to convert all the retrieved P-frames into I-frames at the decoder, thereby breaking the inter-frame dependencies between the P-frames and the I-frames. After the frame syntax conversion and frame reordering, the motion vector swapping approach developed in [13] can be used for backward-play of the new I-B bit-stream. Although this approach does not require the frame memories to store all the decoded frames for the backward-play, it increases the computational complexity and still requires significant storage to store the converted I-frames at the decoder. It also causes drift in the B-pictures since the converted I-pictures are not exactly the same as the original P-pictures. Wee *et al.* [14] presented a method which divides the incoming I-P-B bit-stream into two parts: I-P frames and B-frames. A transcoder is then used to convert the I-P frames into another I-P bit-stream with a reversed frame order. A method of estimating the reverse motion vectors for the new I-P bit-stream based on the forward motion vectors of the original I-P bit-stream as described in [15] is used to reduce the computational complexity of this transcoding process. For B-frames, the motion vector swapping scheme proposed in [13] is used for reverse-play. The transcoding process, however, still requires much computation and will cause drift due to the motion vector approximation [14]. None of the above methods addressed the problem of the extra network traffic and the decoding complexity caused by some VCR functions such as fast-forward/backward play and random-access.

Since network-bandwidth and the decoder-complexity are major concerns in most streaming video applications, in this paper, we investigate effective techniques to implement the full VCR functionality in an MPEG video streaming system with minimum network bandwidth requirement and decoder complexity. We propose to use dual bit-streams at the server to resolve the problem of reverse-play. Based on the dual-bit-stream structure, we propose a novel frame-selection scheme at the server to minimize the required network bandwidth and the decoder complexity. This scheme determines the frames to stream over the networks by switching between the two bit-streams based on a least-cost criterion. We also describe our implementation of an MPEG-4 video streaming system supporting the full VCR functionality.

The rest of this paper is organized as follows. In Section 2, we discuss issues related to the MPEG video streaming with full VCR functionality. In Section 3, we describe our proposed scheme for supporting full VCR functionality with least network resource and decoding effort. Section 4 shows the performance of the proposed method. Section 5 describes our implementation of an MPEG-4 video streaming system with full VCR functionality. Finally, conclusions are given in Section 6.

## 2. MPEG video streaming with full VCR functionality

A block diagram of an MPEG video streaming system is shown in Figure 1. The video streams are compressed using MPEG video coding standards and are stored in the server. The client can view the video while the video is being streamed over the network. In the client machine, a

pre-load buffer is set up to smooth out the network delay jitter. In this paper we discuss the scenario that the video is streaming over the Internet and the full VCR functionality needs to be supported.

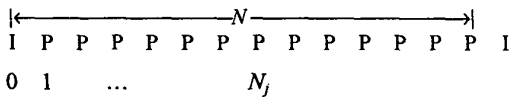
There are many different schemes to encode the MPEG video, depending on the desirable server/network/client complexity requirements. For example, the video can be encoded with all I-frames. This will result in the lowest complexity client machine. However, it will require very large server storage and network bandwidth since the I-frames will result in high bit-rates. Since the network bandwidth usually is the highest concern, we assume that the video is coded with all I-B-P frames that can achieve high compression ratios for the transport over a network with minimum bandwidth resources.

As described in the last section, the trick-modes of the VCR functionality requires higher network-bandwidth and decoder-complexity. In the following, we provide some analyses and simulation results to show how many extra frames need to be sent through the networks and decoded at the client decoder in average. In the analyses, we consider two cases: (1) random access and (2) fast-forward play. Since B-frames are not involved in decoding later frames, for clarity but without loss of generality, we assume the bit-stream contains I- and P-frames only. The results can be easily extended to the I-P-B-frame structure.

### 2.1 Random access

In the random-access operation, the decoder requests a frame with an arbitrary distance from the current displayed frame. If the requested frame is an I-frame, the server side will only need to transmit this frame, and the decoder can decode it immediately. However, if the requested frame is a P-frame, the server needs to transmit all the P frames from the previous nearest I frames to this requested frame.

Suppose all the GOPs in the bit-stream have the same length $N$, and frame $N_j$ is the random-access point.




Then, in order to decode frame $N_j$, frames 0, 1, ...$N_j$-1 should also be sent from the server side. Assuming the random-access points are uniformly distributed, the average number of extra frames to send is $N_{trans}=(N-1)/2$. For example, when $N$=14, $N_{trans}$=6.5, meaning that an average of 6.5 extra frames (a total of 7.5 frames) should be transmitted over the network and decoded by the decoder in order to decode the requested frame in the random-access mode.

### 2.2 Fast-forward play

Suppose frame $N_j$ is the starting point of the fast-forward operation, and $k$ is the fast-forward speed-up factor (e.g. for $k$=6, only one frame will be displayed every 6 frames). Since the next frame to be displayed is $N_{j+k}$, the server may send the frames $N_{j+1}$ $N_{j+2}$ ... $N_{j+k}$, so that totally $k$ frames will be received by the client side to decode the frames $N_{j+1}$ $N_{j+2}$...$N_{j+k}$ (but just displays the frame $N_{j+k}$).

In fact, the server may not need to send so many frames. For example, consider the case:

| | 9 | | | | | 14 | 15 | 16 | 17 | 18 | 19 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ... | P | P | P | P | P | I | P | P | P | P | P | ... |

where frame 9 is the current displayed frame, and frame 15 is the next frame to be displayed under the fast-forward mode ($k$=6). Apparently, there is no need to send frames 10-13, since they are not needed for the decoding of frame 15. Therefore, the server can just send frames 14 and 15. Similar to the random-access mode, the starting point of the fast-forward mode can be any frame in a GOP. Different starting points will lead to different numbers of frames to be sent. We have simulated the situation of $N$=14, $k$=6. As shown in Table 1, the average total number of frames needs to be sent for decoding a displayed frame is about 4.76 frames.

## 3. Supporting full VCR functionality with minimum network bandwidth and decoder effort

To solve the problem in the backward-play operation, we propose to add a reverse-encoded bit-stream in the server. To generate the reverse-encoded bit-stream, in the encoding process, after we finish the encoding and reach the last frame of the video sequence, we encode the video frames in the reverse order. The proposed dual-bit-straem structure is shown in Figure 2, where "$F$" and "$R$" stand for the forward and reverse encoded bit-streams respectively. For clarity of the presentation but without loss of generality, in this paper, we use an example in which the video is coded in I/P-frames with a GOP size of 14 frames as shown in Figure 2. The extension of our discussion to the case with the general I-B-P GOP structure is straightforward.

In Figure 2, we arrange the encoding so that the I-frames in the reverse bit-stream are interleaved between the I-frames in the forward bit-stream. In this way, the required number of frames sent by the server and decoded by the decoder can be further reduced in other trick modes as will be explained later. Alternatively, the I-frames in both streams can be aligned to save storage since the two I-frames in the forward and the reverse bit-streams are the same, and only need to be stored once. Two metadata files

recording the location of the frames in each compressed bit-stream are also generated so that the server can switch from the forward-encoded bit-stream to the reverse-encoded bit-stream and vice versa easily. I-frames represent the access points to switch the streams. With the reverse-encoded bit-stream, when the client requests the backward-play mode, the server will stream the bits from the reverse-encoded bit-stream. Using this scheme, the complexity of the client machine and the required network bandwidth for the backward-play mode can be minimized. The storage requirement of the server will be about doubled. However, this is usually much more desirable than to require a large network bandwidth and to increase the complexity of the client machines since the network bandwidth is more precious and there may be a large number of client machines in the streaming video applications. Since the encoding of the video is done off-line and can be automated, the extra time needed in producing the reverse-encoded bit-stream is not an important concern.

To reduce the decoder complexity and the network traffic in the fast forward/backward and the random-access modes, we propose a frame-selection scheme which minimizes a predefined "cost" using bit-stream switching. The cost can be the decoding effort at the client decoder or the traffic over the networks, or a combination of both. This is further explained as follows.

Let $c_{R\_C}$ stands for the cost of decoding the next requested P-frame from the current displayed frame, $c_{R\_FI}$ for the cost of decoding the next requested P-frame from the closest I-frame in the forward bit-stream, and $c_{R\_RI}$ for the cost of decoding the next requested frame from the closest I-frame of the reverse-encoded bit-stream. To minimize the number of frames sent to the decoder, the costs can be the distances from the possible reference frames to the next requested frame. To minimize the network traffic, the costs can be the total number of bits from the possible reference frames required for decoding the next requested frame. The bit-rate calculation can be done simply by recording the number of bits used for each encoded frame in the metadata file in the pre-encoding process, and summing up the bit-rates of those frames to be sent. In general, a larger number of frames to be sent implies heavier network load. However, it also depends on the numbers of I-, P-, and B-frames to be sent since the numbers of bits produced by these three types of frames vary greatly. It is also possible to use different weights to combine the two costs according to the channel condition and the client capability. Based on the current play-direction, the requested mode, and the costs $c_{R\_C}$, $c_{R\_FI}$, and $c_{R\_RI}$, the reference frame to the next requested frame with the least cost will be chosen to initiate the decoding. This will also determine the selection of the next bit-stream and the decoding direction. This least-cost criterion will only be activated in the fast forward/backward and the random access modes to avoid frequent bit-stream switching in the normal forward/backward operations.

To illustrate the scheme, we use the example in Figure 2 again, assuming that the previous mode was backward-play and the requested mode is fast-backward with a speed-up factor 6 which needs to display a sequence of frames with frame-numbers 20, 14, 8, 2, .... For simplicity, in the following examples we use the minimum decoding distance criterion (which roughly corresponds to minimum decoder complexity) to illustrate the selection of the next reference frame and the effectiveness of the proposed method.

Our method will operate as follows:

1. The current position is frame 20 which was decoded using the reverse bit-stream (*R*).
2. Frame 14 will be decoded from the forward bit-stream (*F*) directly since it is an I-frame.
3. Frame 8 will be decoded from frame 7 of the backward bit-stream, since the distance between frame 7 of the reverse bit-stream (an I-frame) and the requested frame (frame 8) is less than the distances between the requested frame and the current decoded frame (frame 14 of the reverse bit-stream), and the closest I-frame of the forward bit-stream (it's also frame 14). Note that, in this case, we use frame 7 of the reverse bit-stream (an I-frame) as an approximation of frame 7 of the forward bit-stream (a P-frame) to predict frame 8 of the forward bit-stream. This will cause some drift. However, in the fast-forward/backward modes, the drift is relatively insensitive to human eyes due to the fast change of the content displayed. When it resumes normal forward/reverse play, the I-frames will terminate the drift. The drift problem will be further investigated in the next section.
4. Frame 2 will be decoded from frames 0 and 1, using the forward bit-stream, since the decoding effort from frame 0 of the forward bit-stream (an I-frame) is the minimum.

The bit-stream sent from the server will have the following form:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **P** | **I** | I | **P** | I | P | **P** | ... | frame type |
| **20** | **14** | 7 | **8** | 0 | 1 | **2** | ... | frame number |
| ***R*** | ***F*** | *R* | ***F*** | *F* | *F* | ***F*** | ... | selected bit-stream |

The frames indicated by the bold-face are those to be displayed at the client side. In this way, we only need to send and decode 6 frames. Without the minimum effort decoding scheme, we will need to send and decode 13 frames from the reverse bit-stream.

In the case of random access, frame-jump will be performed followed by normal forward-play. For example, the client requests random access to frame 22 when the current decoded frame is frame 3. With the proposed

method using the minimum decoding distance criterion, the server streams the bit-stream as follows:

| | | | | | |
|---|---|---|---|---|---|
| **P** | **I** | **P** | **P** | **P** | ... frame type |
| **3** | **21** | **22** | **23** | **24** | ... frame number |
| ***F*** | ***R*** | ***F*** | ***F*** | ***F*** | ... selected bit-stream |

In this example, we only need to send and decode 2 frames to reach frame 22. Without our proposed least-cost scheme, it will require to send and decode 9 frames from frame 14 (an I-frame) using the forward bit-stream. Again, in this example, for frame 21, we use the I-frame in the reverse bit-stream to approximate the P-frame in the forward bit-stream. This will cause drift but the drift will only last a few frames within the GOP (a fraction of a second) since the video content will be refreshed by the I-frame in the next GOP. Thus it should not be a problem.

If the minimum decoding distance criterion is used (i.e., to minimize the number of frames sent to the decoder), the proposed scheme will guarantee that the maximum amount of decoding to access any frame in the sequence is less than GOP/4 frames if the I-frames in the forward and the reverse bit-streams are interleaved. In addition, no huge temporary buffer is required in the decoder. If the I-frames in the forward and the reverse bit-streams are aligned, the maximum amount of decoding to access any frame will be less then GOP/2 frames.

## 4. Performance analysis of the proposed dual-bit-stream least-cost method

In the following, we analyze the performance of the proposed method using the minimum decoding distance criterion for the random access and the fast-forward play modes.

### 4.1 Random access

Using the dual-bit-stream example shown in Figure 2, we label the frames in the GOP where the requested frame lies as 0, 1, 2, ..., $N$-1, frame $N_j$ is the random access point. $N_{RI}$ is the position of the I-frame in the reverse bit-stream.

In this case, the frames to be transmitted for decoding frame $N_j$ will be decided by two distance measures, one is the distance from frame $N_j$ to the nearest I frame in the forward bit-stream, and the other distance is from $N_j$ to the nearest I-frame in the reverse bit-stream.

Assume $N_j \in [0,N-1]$, $N_{RI}$ is in the range of [1,N-1]. For simplicity but without loss of generality, we assume that $N$ is even and $N_{RI}$ is odd as in our previous example. We can observe that:

- The minimum number of extra frames to be sent via the network is 0 (when $N_j$=0 or $N_j$=$N_{RI}$);
- The maximum number of extra frames to be sent via the network is

$$\max\left(\frac{N_{RI}-1}{2}, \frac{N-1-N_{RI}}{2}+1\right);$$

- The average number of extra frames to be sent via the network is:

$$N_{trans} = \sum_{i=0}^{\frac{N_{RI}-1}{2}} \frac{i}{N} + \sum_{i=\frac{N_{RI}+1}{2}}^{N_{RI}} \frac{(N_{RI}-i)}{N} + \sum_{i=N_{RI}+1}^{\frac{N_{RI}-1+N}{2}} \frac{(i-N_{RI})}{N} + \sum_{i=\frac{N_{RI}+1+N}{2}}^{N-1} \frac{(N-i)}{N}$$

$$= \frac{2N_{RI}^2 - 2NN_{RI} + N^2 - 2}{4N}$$

By taking the derivative with respect to $N_{RI}$, we can find that when $N_{RI}$ takes the odd number closest to $N/2$, $N_{trans}$ can take the minimum value of

$$N_{trans} = \begin{cases} \frac{N}{8} - \frac{1}{2N} & (\frac{N}{2}=\text{odd}) \\ \frac{N}{8} & (\frac{N}{2}=\text{even}) \end{cases}$$

For example, when $N$=14, $N_{RI}$=7, $N_{trans}$=1.72, meaning that an average of 1.72 extra frames (i.e., 2.72 total frames) should be transmitted to decode 1 frame in the random access mode. Apparently, this is much better than the case in Section 2 (7.5 total frames without our scheme).

### 4.2 Fast-forward play

We simulated the situation of $N$=14, $N_{RI}$=7, $k$=6. Table 2 shows the simulation result. The average total number of frames to be send before decoding a requested frame is 2.71 (compared to 4.76 without our scheme).

Figure 3 compares the average numbers of frames sent to the decoder for decoding a frame and the average network traffics with and without the proposed dual-bit-stream least-cost method with respect to different speed-up factors in the fast-forward operation. Two bit-streams generated by forward and reverse encoding a 280-frame (20 GOPs in our example) "Mobile and Calendar" test sequence at 3 Mbps with a frame-rate of 30 fps were used for simulation. From the above analyses, in the fast-forward/backward and random-access operations, the sever needs to send several extra frames to the decoder to display one frame, thereby resulting in a heavy burden on the networks (especially when the number of users is large) and increasing the decoder complexity. Note that, with the proposed method, when the speed-up factor reaches GOP/4 (e.g., 3.5 in our example), the decoding complexity and the network traffic will not continue to grow even when the speed-up factor gets higher. This is because, when the speed-up factor $k$ is larger than GOP/4, the server always can find an I-frame in one of the two bit-streams which has shorter distance to the next displayed frame from the current displayed P-frame, since the distance for the nearest I-frame is guaranteed to be less than GOP/4. In this case

the number of frames to be sent for displaying a requested frame will have range of [1, GOP/4+1]. If the distribution is uniform, the average number of frames can be approximated by GOP/8+1, which is 2.75 when GOP=14, very close to the simulation result. From Figure 3, it is obvious that the proposed method can achieve significant performance improvement in terms of the decoder complexity and the network traffic load. When the speed-up factor $k \geq$ GOP/4, the proposed method guarantees a nearly constant decoding and network traffic cost.

## 5. Implementation of an MPEG-4 video streaming system with full VCR functionality

We have implemented an MPEG-4 [12] video streaming system to demonstrate the effectiveness of our scheme. Figure 4 illustrates the overall structure of the system and depicts its major functional blocks.

The client station is connected to the remote video server over an IP network and requests access to a compressed video sequence. Two logical channels are established between the server and the client: the data channel and the control channel. The server delivers the requested MPEG-4 bit-stream through the data channel and receives VCR commands through the control channel.

As shown in Figure 4, the video server consists of a VCR Manager, a Bit-stream Manager, an MPEG-4 System Network Server, and a Video Database. The client station consists of an MPEG-4 Player, an MPEG-4 Object Decoder, a Scene Composer, and an MPEG-4 System Network Client. The client station will access the video server and interactively retrieve a video sequence through the Graphic User Interface of the MPEG-4 Player.

The MPEG-4 Player provides the user interface and displays video frames sent by the server according to the user's requests. According to the specific VCR function that the user selected through the user interface, the Player generates the requested frame-number and sends it to the MPEG-4 System Network Client. The MPEG-4 Object Decoder receives the corresponding bit-stream from the MPEG-4 System Network Client, performs the decoding procedure of the received bit-stream, and transfers the decoded frames to the MPEG-4 Player.

The MPEG-4 System Network Server manages the network connection. The Bit-stream Manager maintains the record of the video bit-streams, and the VCR Manager handles the state-machine described in the previous section.

## 6. Conclusions

In this paper, we discussed issues in implementing an MPEG video streaming system with full VCR functionality. We showed that when the users request reverse-play, fast-forward/reverse-play, or random access, it may result in much higher network traffics than the normal-play mode. These trick-modes may also require high client machine complexity. We proposed to use a reverse-encoded bit-stream to simplify the client terminal complexity while maintaining the low network bandwidth requirement. We proposed a minimum-cost frame-selection scheme which can minimize the number of frames needed to be sent over the network and to be decoded. We also described our implementation of an MPEG-4 video streaming system. We showed that with our proposed scheme, an MPEG-4 video streaming system with full VCR functionality can be efficiently implemented to minimize the required network bandwidth and decoder complexity.

## 7. References


[1] M. S. Chen, D. D. Kandlur, "Downloading and stream conversion: supporting interactive playout of videos in a client station," *Second Int. IEEE Conf. Multimedia Computing and Systems*, pp. 73-80, Washington, 1995.

[2] T. D.C. Little and D. Venkatesh, "Prospects for interactive video-on-demand," *IEEE Multimedia*, vol. 13, pp. 14-24, Aug. 1994.

[3] Microsoft Windows Media, Microsoft Corporation Inc., http://www.microsoft.com/windows/windowsmedia/

[4] Apple QuickTime Player, Apple Corporation Inc., http://www.apple.com/quictime/

[5] Real Networks RealPlayer, http://www.real.com/

[6] Relay Networks ReplayTV, http://www.replay.com/

[7] TiVo Inc., http://www.tivo.com/

[8] D. Wu *et al.*, "Transporting real-time video over the Internet: challenges and approaches," to appear in *Proc. IEEE*, 2000.

[9] F. C. Li *et al.*, "Browsing digital video," Technical Report: MSR-TR-99-67, Microsoft Research, Sep. 1999, ftp://ftp.research.microsoft.com/pub/tr/tr-99-67.pdf

[10] ISO/IEC 11172, "Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbits/s," (MPEG-1), Oct. 1993.

[11] ISO/IEC 13818-2 "Generic coding of moving pictures and associated audio". (MPEG-2), Nov. 1993.

[12] ISO/IEC JTC1/SC29/WG11 "Coding of moving pictures and associated audio MPEG98/W2194," (MPEG-4), Mar. 1998.

[13] S. Chen, "Reverse playback of MPEG video," U.S. Patent 5,739,862.

[14] S. J. Wee and B. Vasudev, "Compressed-domain reverse play of MPEG video streams," *Proc. SPIE Conf. Multimedia Syst. and Appl.*, pp. 237-248, Nov. 1998.

[15] S. J. Wee, "reversing motion vector fields," *Proc. IEEE Int. Conf. Image Proc.*, Oct. 1998.

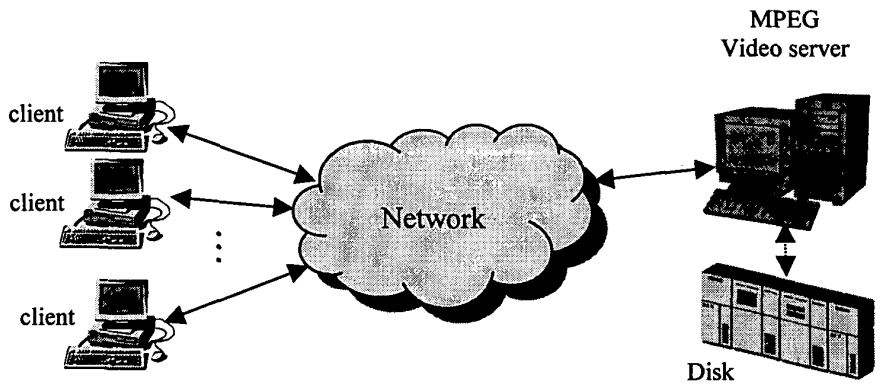



Figure 1. MPEG video streaming.

Table 1. Average total number of frames transmitted for each displayed frame in the fast-forward mode with respect to different starting points using the traditional method ($N$=14, $k$=6)

| Start Point | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Frames Transmitted | 4.6 | 5 | 4.6 | 5 | 4.6 | 5 | 4.6 | 5 | 4.4 | 4.8 | 4.5 | 4.9 | 4.6 | 5 |
| Average total number of frames transmitted to display one frame: 4.76 | | | | | | | | | | | | | | |

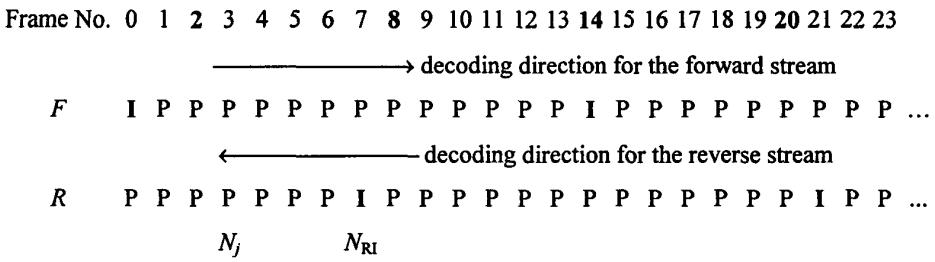
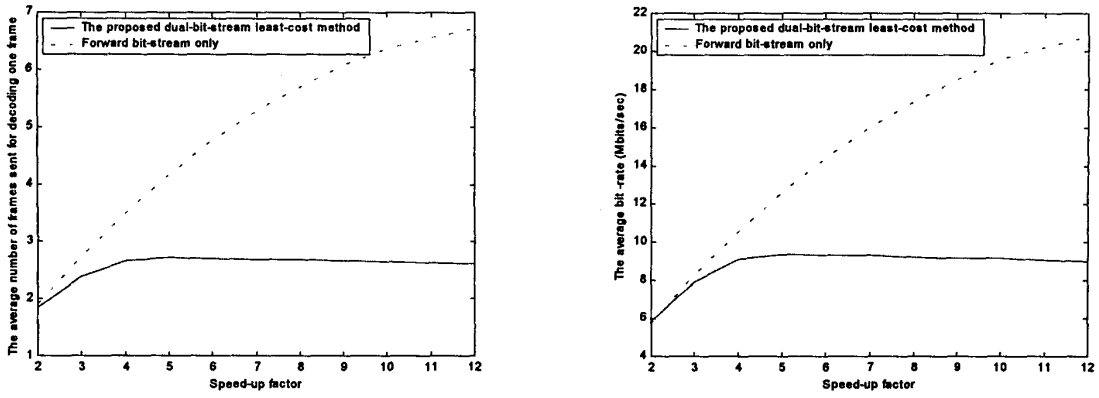



Figure 2. The proposed dual-bit-stream structure.

Table 2. Average total number of frames transmitted for each displayed frame in the fast-forward mode with respect to different starting point using the proposed method ($N$=14, $k$=6)

| Start Point | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Frames Transmitted | 2.7 | 2.7 | 2.6 | 2.7 | 2.7 | 2.8 | 2.7 | 2.7 | 2.6 | 2.7 | 2.7 | 2.8 | 2.7 | 2.8 |
| Average total number of frames transmitted to display one frame: 2.71 | | | | | | | | | | | | | | |




(a) (b)

Figure 3. (a) Average numbers of frames; (b) average bit-rates to send the "Mobil and Calendar" sequence over network with respect to different speed-up factors with (the solid lines) and without (the broken lines) the proposed method.

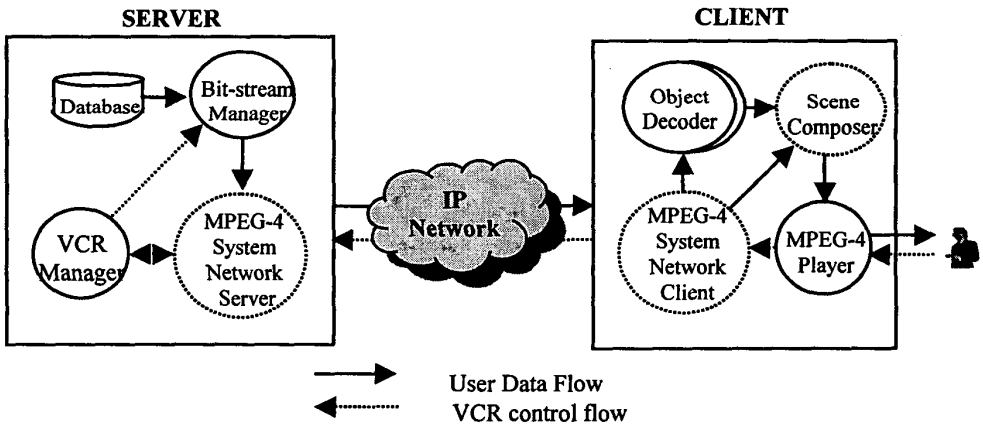



Figure 4. System architecture of the proposed MPEG-4 video streaming system.