To appear in Graphic Communication Series (Special Issue on The 1st International Conference on Graphic Communications), Wuhan, China, May 2006.

A Video Reconstruction Approach Supporting Frame Skipping in H.264

Karl. R.P.H. Leung [a], Everest K.W. Kwok [b], W.K. Chan [c]

[a] Information & Communications Technology Department, Institute of Vocational Education (Tsing Yi), Hong Kong ([a]kleung@vtc.edu.hk);

[b] Department of Computer Science, The University of Hong Kong, Pokfulam Road, Hong Kong ([b]everestkwok@gmail.com)

[c] Department of Computer Science, Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong ([c]wkchan@cs.hku.hk)

ABSTRACT

Mobile real-time video streaming for web-casting and video conferencing conforming to the ITU-T H.264 data compression recommendations format is emergent. However, the quality of the video services for typical applications is hard to foreseen and sustain. Qualities of Services (QoS) approaches, such as video frame skipping techniques, are thus attractive to address the problem. Many conventional video frame-skipping techniques focus on the system-centric quality attributes to adjust the quality of video delivery. They normally ignore the user-centric quality attributes. A user-centric approach, amongst others, is to take human perception in consideration and adapt the video quality expressed in patterns of Group of Pictures (GoP). In a post-processing step of an encoding stage, it filters and re-titles the quality-matching patterns from the rest. It then reconstructs the filtered GoP according to the H.264 format. Most existing H.264 playback engines unfortunately cease to work properly for video steaming with skipped frames. This paper reports our experience on alleviating the playback problem through a case study on our effort to develop a prototype of such a playback engine. It studies how to reconstruct a video steaming in the H.264 format on the fly in the presence of predictable skipped frames, satisfying certain selected user-centric QoS quality levels. We also analyze the applicability of the approach.

Keywords: H.264, frame skipping, QoS control, QoS-human, video reconstruction, human perception

1. INTRODUCTION

In mobile applications such as web-casting and video conferencing, real-time video streaming compliant to the ITU-T H.264 data compression recommendations format [8] is emergent. Roughly speaking, a video is logically a sequence of pictorial frames, for a *frame-based video playback engine*, which we call it an *application* in this paper, may display its motion contents on a screen. A user would perceive a movie from a particular sequence of pictorial frames only if they are in proper timely order; otherwise, the user may consider the video in a poorer playback quality. Since an application may control the level of clarify (e.g., resolution) of a video for its users, it is feasible to improve the playback quality by a two phase processes. The application firstly skips certain pictorial frames, which chiefly aim to reduce the impacts of, for example, network congestion, frame corruptions and lacking in local resources. It then detects and restores the skipped frames from the remaining ones. They open an approach to controlling the Quality of Services (QoS) in real time video streaming applications [1, 2, 3]. Applications having the above two properties are increasingly popular.

The application skips certain pictorial frames for handling various situations at the source site, during the transmission and at the destination site. We give a few examples as follows. At a source site, a server program may fail to send frames for the reasons of heavy loading. Even the server program sends all intended frames; the open network may drop or corrupt some of these frames. At a destination site, certain sequences of frame may arrive not in order or miss the timing requirements of the movie that they constitute, even though they have arrived.

Our previous work [1] proposes a type of user-centric QoS control over open network, which indicates via a control experiment that, in terms of human perception, an application may skip additional frames to achieve a user-centric QoS level similar to one without skipping these additional frames. The result is promising in the sense that a server program may transmit fewer frames to a client program if the server program knows such QoS requirements. This lower the requirements on the bandwidth and compromises no QoS requirements distinguishable from the human perspective. It assumes the application able to play back a video in which certain frames that are intentionally discarded, which we term them as *QoS-irreverent frames*.

Liu and Kender [10] address the problems of video reconstruction by selecting key frames having low error rate. Isovic

et al. [11] propose to take preventive and reactive measure to minimize frame skipping. We [1] skip frames intentionally.

Unfortunately, many popular video playback engines [4, 5, 6] cease to work when they detect any QoS-irreverent frames. To evaluate our proposal, we develop our own middleware-based solution, which plays back videos with QoS-irreverent frames [1]. The major challenge of the development is to reconstruct a video in real time. This paper reports the performance evaluation.

The rest of the paper is as follows: Section 2 provides the background on the problem. Section 3 firstly overviews our previous proposal followed by an experiment to evaluate the proposal. The performance of the proposal is discussed in Section 4. Section 5 discusses the advantages and limitations of the proposal as well as concluding this paper.

2. PROBLEM ANALYSIS

2.1 Frames dependency in H.264

H.264 has made use of several video compression techniques such as bidirectional motion compensation [7], multi-frame frame compensation [7] and superimposed frame compensation [8], to improve its compression rate. These techniques compare frames in a Group of Picture (GoP), using the video objects of the frames, to derive object dependencies amongst these frames.

Generally, a GoP is a sequence of video frames. In the H.264 format, there are three basic types of frame, namely B-frame, P-frame and I-frame. A B-frame depends on the two P- frames that appear ahead of and behind the B-frame. Similarly, a P-frame depends on a combination of I-frames and P-frames (which is denoted by I/P frames) that appears ahead of and behind the P-frame. An I-frame is a standalone frame, and hence, it is independent to other frames.

Modern video coding algorithms usually reuse objects to enhance coding efficiency [9]. For example, a GoP scheme in MPEG-4 may strictly define the number of frames of each types and their structural relationships in a GoP. Since all GoP of a video resembles structurally one another, using these common properties in video encoding will usually achieve a better compression rate. Skipping a QoS-irreverent frame in a GoP will destroy the above similarity however, which in turn may mislead applications in performing video playbacks.

2.2 Video playback mechanism

In this section, we describe the video playback mechanism. An application may preload upcoming video contents into its processing memory before displaying the contents on screen. After the preloading process, the application normally validates the format and the completeness of the incoming video streams. Based on the validation results, it either starts playing the video successfully, or denies playing it if the validation fails. There are specific parameters to locate the header or the trailers of a frame within a GoP. When frames of a GoP are skipped abnormally without a suitable reconstruction process, the structure of the GoP becomes invalid, leading popular applications [4, 5, 6] cease to work.

Specifically, the general ceasing situation is as follows. After the application starts playing a video, it looks up its codec library and selects suitable coding protocols to decode the video. When certain specific frame are missing, the video player will terminate. It is because certain missing frames would affect the structure of GoPs, resulting in irregular GoP patterns within a video, violating the parameters specified in the video header.

2.3 Frame skipping methods

Frame skipping is an approach to reduce the number of frames, which are sent to an application for video streaming activity. We describe two methods in this section.

Time (ms)	33	66	99	132	165	198	231	264	297	330	363	396	429	462	495	528	561	594	627	660	693	726	759	792
Frame	I	B	B	P	B	B	P	B	B	P	B	B	I	B	B	P	B	B	P	B	B	P	B	B
(after encoding to skip frames)																								
Frame	I	X	B	X	B	X	P	X	B	X	B	X	P	X	B	X	B	X	P	X	B	X	B	X

Table 1. Frame skipping method 1.

A popular approach is to reduce the total number of frames without altering the encoding configuration in a pre-processing encoding stage. Table 1 shows an example. The first row of the Table 1 shows a video having 120 frames with 29.97 frames-per-second (fps), lasting for 4 seconds. Its GoP pattern is “IBBPBBPBBPBB” as shown in the second

row of Table 1. Suppose that we drop 50% of all frames without violating the original "IBBPBBPBBPBB" pattern. The resultant total number of frames will be reduced to 60 (The third row of Table 1, in which an 'X' means a skipped frame in the location concerned). Since some frames are dropped, the playback duration of the video will be shortened. This skipping frame approach normally requires some pre-processing and incurs heavy computation overhead.

Another approach is to skip frames according to the QoS-Human quality attribute [1] which is determined by a priori (pilot) user session. Table 2 shows the same video clip, in which the GoP pattern is changed into "IXXPXXPXXPBB". Furthermore, in our control study, since the GoP patterns vary dynamically according to the QoS requirements of individual users [2, 3], predetermining a configuration parameter applicable to all cases is infeasible. This results in a conflict between the received GoP pattern and the configuration parameter, which cease the concerned application to playback the video. Hence, we introduce a mechanism [2, 3] to make the application to play back the received frames disregard of the GoP patterns. Our approach [2, 3] will not alter the playback duration, as this would have unpredictable impact on the associated QoS.

Time (ms)	33	66	99	132	165	198	231	264	297	330	363	396	429	462	495	528	561	594	627	660	693	726	759	792
Frame	I	B	B	P	B	B	P	B	B	P	B	B	I	B	B	P	B	B	P	B	B	P	B	B
(after directly frame skipping)																								
Frame	I	X	X	P	X	X	P	X	X	P	B	B	I	X	X	P	X	X	P	X	X	P	B	B

Table 2. Frame skipping method 2.

3. OUR SOULTION FOR FRAMES RECONSTRUCTION

In this section, we first refine our preliminary idea to address missing frame, which were reported in [2, 3]. Next, we describe the setup of the experiment to observe the performance characteristics. The result of our performance experiment will be presented in section 4.

3.1 Our approach

Our preliminary idea [2, 3] is to substitute a missing frame (due to whatsoever reason) by a dummy frame. The technical difficulties, such as player ceasing to work, are not addressed.

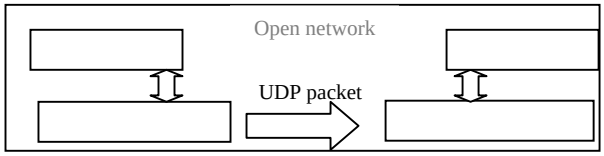
In this paper, we exploit the concept of using blank frame having the same type of the skipped frame. In other words, we realize that a dummy frame contains the start tag and end tag of the required frame type, as well as a frame body that contains the same number bytes of the skipped frame size. Each byte in the frame body is set to zero in value. This strategy will reconstruct a GoP pattern conformed to the configuration parameters. Therefore, it resolves the problem of the video players ceasing to operate. As the number of frames remains unchanged, ideally, the playback duration would only be affected marginally. In the next section, we describe an experiment to evaluate the performance of our approach. To ease our discussion, in the sequel, we use the term *video player* to mean an application.

3.2 Experiment

To testify our approach without re-developing a video server or a video player, we also proposed to use a middleware-based approach. We have developed a middleware [2, 3] to link up a video server and video players. We install the video server and the video players on different machines. We run them separately on two independent machines connected with Ethernet network. The server is on a more powerful machine (Pentium 4 CPU 2.6GHz 1.00GB RAM) and the client is on less powerful machine (Pentium 4 PCU 1.0GHz 512MB RAM). The middleware consists of a server-ware and some client-ware. The server-ware acts as a gateway of the video server, responsible for dropping frames with reference to both the QoS-Human quality attribute [1] and the best QoS quality that it can offer to individual video players. Every video player at a client side communicates with a client-ware. The client-wares are responsible to feed back the QoS requirements of its associated video player clients to the server-ware. We use UPD-based open network as the communication system between the server side (i.e. the video server) and the clients (i.e. the video players). Figure 1 shows the architecture of the middleware. The detail is as follow.

In a typical real time video streaming, packets are usually divided into a large number of small packets. We also adopt this strategy in the experiment. We consider a video as a sequence of bits. For different video coding standards, although coding methods may differ, yet their encoding format pinpoints the start flags or the end flags of interest video objects in the stream of bits. Based on these flags, one can identify the nature of video objects and help organize video segment

into packets.



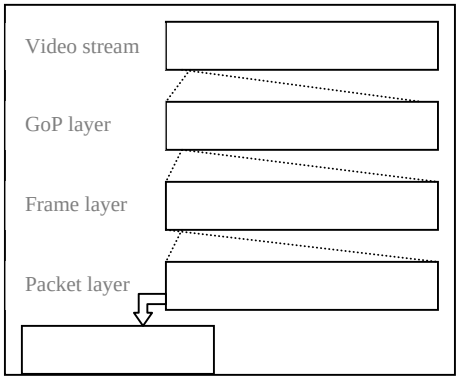


Figure 1. Frame skipping approach architecture

Figure 2. The origin of each packet from video stream

As shown in Figure 2, every packet has its specific source of origin (such as video, GoP number, frame number or packet number). This information uniquely identifies a package and is stored in the packet header document of the packet. We do a similar process at every layer. After the necessary packing of packets, they will be sent to their corresponding destinations. The packet header format is described in Table 3.

Name	Size (bit)	Description
Video id	8	The signature of video stream
GoP #	16	The number of GoP sending
Frame #	8	The number of frame within GoP sending
Packet #	16	The number of packet within frame sending
Timestamp	32	The record of packet sending time
Control flag	8	The flag to pass message between server and client

Table 3. The format of our packet header

On a client side, the client-ware receives packets and groups them into pictures, which form GoPs. When the client-aware detects any missing frame, it generates a dummy frame according to the skip frame type and the frame size. The dummy frame is then inserted into the GoP as if it is an ordinary frame. Finally, an entire re-constructed GoP will be passed to the associated video player to playback. Figure 3 depicts the process.

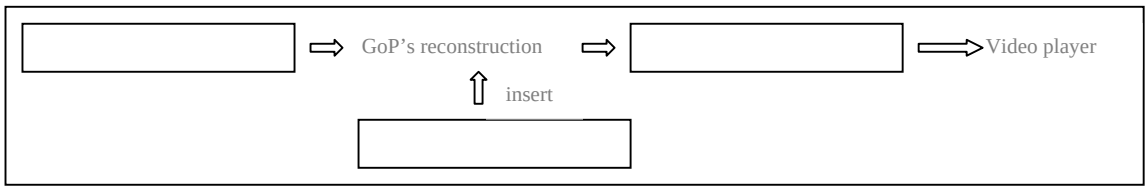


Figure 3. Dummy frame inserting into an incoming GoP with skipped frames

4. EXPERIMENTAL RESULT

4.1 Determining GoP patterns for QoS-Human

Our approach needs to firstly determine a GoP pattern based on the QoS-Human quality metric. We run the experience on three types of video namely stationary, carton and action, which have twelve frames per GoP. They all last for 60 seconds and have the GoP pattern "IBBPBBPBBPPP". Their video clip sizes are 18.6, 24.8 and 28.3 MB, respectively. We display the video at the 800×600 resolution mode. We select GoP pattern that scores highest in the experience session the QoS-human. The chosen GoP pattern is "IXXPXXXXXXXP".

After obtaining a desirable set of GoP pattern, we proceed to measure the performance of our middleware-based proposal. The

performance result is shown in the next section.

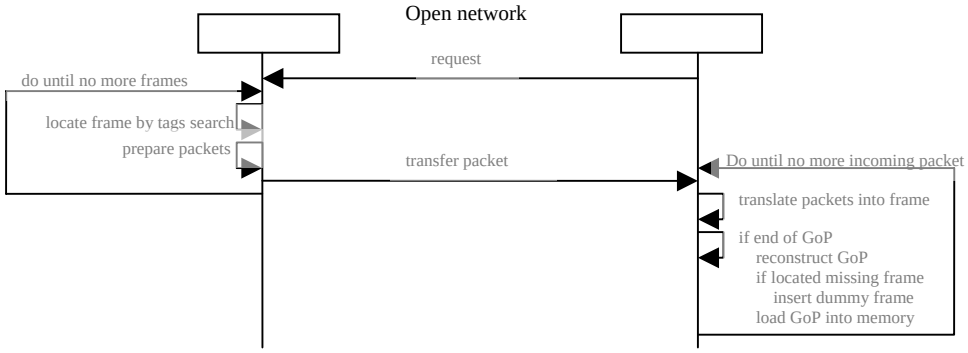


Figure 4. A flow of reference experience

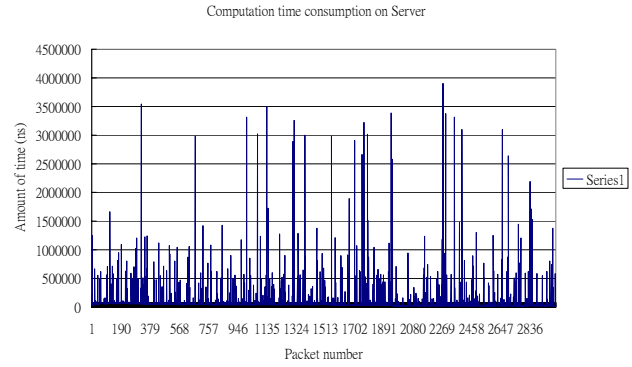


Figure 5. Computation time consumption on Server

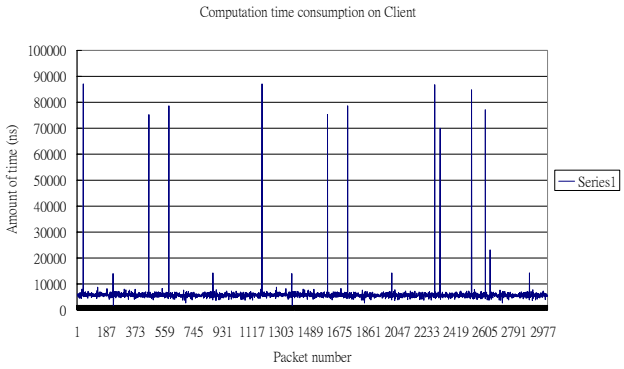


Figure 6. Computation time consumption on Client

4.2 Performance

The aim of the experience is to observe the computation time overhead from the video reconstruction approach so the impact should be short. Concerning the impact on performance, the server-ware has to conduct an exhaustive search to locate the start and the end flags of a frame, and then to prepare packets for the transmission purposes. We also recall that, on a client side, a client-ware receives packets, translating them into frames, and generating necessary dummy frames to pretend them as the skipped frames within GoPs. In other words, extra computation efforts have spent on the frame selection, the packet formation and the dummy frame generation. They will result in some delay at both the server and clients.

The running result on three video shows similar patterns, hence we report one for space restriction. In Figures 5 and 6, we show the computation time consumption with the subject video (stationary) on the server and client sides, respectively. In Figure 5, the workload of the server is not evenly distributed. Apart from transmitting the video contents

as a typical server does, the server-ware (also on the server machine) is responsible to conduct an exclusive search on every frame to skip frames. Since frames may have different sizes, the time to prepare GoPs will differ. There are some sharp "jumps" on the process time consumption on server side. We observe that it is mainly due to the server-ware needing to search frames and preparing to send packages at the same time. From Figure 5, the peaks may reach 4 seconds. On average, it only doubles the workload of the server if the video server runs without the server-aware. The result is promising as it indicates that our approach can easily be implemented by doubling the hardware requirements, instead of an exponential increase.

In Figure 6, the workload of a client distributes more evenly and lighter. However, it is similar to the performance pattern of the server side that some occasional jumps. We observe that it is mainly due to the time crash to construct a GoP and to load GoP into memory. It is encouraging as jumps are occasional instead of regular. We observe that the performance of a client is only marginally affected.

5. DISSCUSSION AND CONCLUSION

In this paper, we have proposed an approach to substituting a missing frame by a dummy frame, and realized the proposal by a middleware-based approach. The results of the experiment show that if there are missing frames, playback will not cease if we appropriately insert dummy frame to pretend missing frame. This allows clients to reconstruct video in presence of intentionally skipped frame. Moreover, it indicates that it is feasible to use user-centric quality attribute in real time environment, i.e., it can apply QoS human quality attributes [1]. However, extra processing resources are required at both the server and client. It also indicates that the uneven patterns of time consumption affect the overall performance of system especially in server. We plan to investigate other efficient re-construction techniques in future.

We have proposed a simple and effective approach to reconstructing GoP with missing frames, which substitute any missing frame by a dummy frame of the same frame type. This approach helps the video players to continue playing the video even though some frames are missing or incompatible to the pre-defined configuration. We have also proposed a middleware-based technique so that our proposal may augment other work. The performance experiment reported in this paper indicates that it is feasible to implement our re-construction approach to support real time video streaming services.

REFERENCES

1. Joseph K.Y. Ng, Karl R.P.H. Leung, Wai Wong, Victor C.S. Lee and Calvin K.C. Hui. "A scheme on measuring MPEG video QoS with human perspective", In *Proceedings of the 8th International Conference on Real-Time Computing Systems and Applications* (*RTCSA 2002*), Tokyo, Japan, pp. 233–241, March 18–20, 2002.
2. Karl R.P.H. Leung, Joseph Kee-yin Ng and Calvin K.C. Hui. "QoS-Aware middleware for MPEG video streaming", In *Middleware for Communications*, Qusay H. Mahmoud (editor), John Wiley & Son, ISBN 0-470-86206-8, pp. 331–357, 2004.
3. Joseph K.Y. Ng, Wai Wong, Calvin K. Hui and Karl R.P.H. Leung, "The Implementation of a multi-server distributed MPEG video system", In *Proceedings of the 7th IEEE Real Time Technology and Application Symposium* (*RTAS 2001*), Taipei, Taiwan, pp. 111–113, May 31–June 2, 2001.
4. Microsoft Corporation. "Microsoft Windows Media Player 10". Available at: http://www.microsoft.com/windows/windowsmedia/default.mspx (last accessed: February 2006).
5. RealNetworks. "RealPlayer version 10.0 for Windows". Available at: http://www.real.com/international/player/?src=hk_realhome&lang=tw&loc=hk (last accessed: February 2006).
6. Elecard. "Elecard MPEG Player 4.0.2". Available at http://www.elecard.com/products/products-pc/consumer/144/mpeg-player (last accessed: February 2006).
7. Markus Rlierl and Bernd Girod. *Video coding with superimposed motion compensated signals application of H.264 and beyond*, Kluwer academic publishers, pp 1–33, 67–104, 2004.
8. Peter Symes. *Digital video compression*, McGraw-Hill, 2001, pp 137–221.
9. Roger Finger and Andrew W. Davis. "Measuring video quality in videoconferencing systems". Available at: http://www.wainhouse.com/articles/article_j.html (last accessed: February 2006).
10. Tiecheng Liu and John R. Kender. "An Efficient Error-Minimizing Algorithm for Variable-Rate Temporal Video Sampling", *In IEEE Proceedings of International Conference on Multimedia and Expo (ICME 2002)*, pp 413–416, 2002. Naples, Italy, 3-5 July 2002.
11. D. Isovic, G. Fohler, and Liesbeth F. Steffens. "Timing constraints of MPEG-2 decoding for high quality video: misconceptions and realistic assumptions" *In Proceedings of the 15th Euromicro Conference on Real-Time Systems (ECRTS 2003)*, pp 73–82, 2003. IEEE Computer Society Press, Washington, DC, USA.