International Journal of Science and Engineering Vol.3, No.3 (2013): 15-21

Design and Implementation of H.264 Player for Multiple Description Coded Videos

Chow-Sing Lin* and Jhe-Wei Lin
Department of Computer Science and Information Engineering National University of Tainan
33, Sec. 2, Shu-Lin St, Tainan 700, Taiwan (R.O.C.)

Abstract -- Scalable video coding techniques, such as multiple description coding (MDC), not only effectively address the issue of delivering videos over error-prone networks but also provides solutions of servicing scalable videos for heterogeneous clients. When multiple description coded video chunks are received at a client, they must be buffered based on network condition and also be timely decoded and merged in order to provide smooth playback. In this paper, we present the design and implementation of a novel H.264 player for multiple description coded videos. There are two major buffers in our MDC player, which are decoded chunks buffer and merged chunks buffer. To avoid buffer overflow or underflow, merging control and transmitting rate control mechanisms are proposed. Experiment results show that with the aids of proposed control mechanisms, no buffer overflow or underflow was experienced during the playback. Most importantly, based on our measurement, the overheads of decoding a chunk and merging chunks are negligible compared to rendering time of one chunk, which would not affect the smoothness of playback at all.

Index Terms -- Multiple Description Coding, Streaming, Scalability, H.264, Buffer Management..

I. INTRODUCTION

To effectively address the issue of delivering videos over error-prone networks, many multiple description coding (MDC) [1]-[5] schemes have been proposed in the last few years. These MDC schemes aim to reduce the effects of transmission errors by coding video signals in two or more independent sub-streams, called descriptions. Descriptions are separately transmitted over separate channels by exploiting path diversity of the multicast tree on peer-to-peer (P2P) networks [6]-[12]. When all descriptions arrive at a receiving side without errors, the fidelity of a video is maintained. In cases of transmission error due to broken links or a substantial data loss, any arbitrary numbers of received descriptions can be combined and decoded to obtain certain levels of video fidelity.

In addition, MDC can also provide scalable Quality of Service (QoS) for heterogeneous clients with various downloading/forwarding bandwidth. The perceived QoS of watching videos depends on the number of descriptions a client receives. Any subset of descriptions can be received and decoded by a peer to reconstruct the original video content with fidelity commensurate to the number of descriptions it is capable of receiving. The more descriptions are received, the higher the quality the reconstructed video content is. The flexibility of MDC enables peers with heterogeneous downloading capacity to receive a limited number of descriptions, but still enjoy a limited level of viewing quality according to the individual downloading capacity [4], [13], [14].

There has been a lot of research into the realization of MDC technique with existing coding standards. In particular, a great deal of effort has gone into developing an H.264/AVC standard-compliant multiple description (MD) encoders, such as Sub-sampling Multiple Description Coding (SMDC) [15], Multiple Description Motion Coding (MDMC) [16], Multiple State Video Coder (MSVC) [17], and the Motion-Compensated Multiple Description (MCMD) [18] video coding. However, there is no significant work to elaborate the design and implementation of decoder for MD coded videos. In our previous work [19], we used SMDC to encode a video source into multiple descriptions in preprocess phase and studied the feasibility of the proposed Loopback-MDC scheme on the streaming system on P2P-CDN network, named TuBeck. In the SMDC, each input frame is sampled along its rows and columns with a sampling factor of 2. Let $x(i, j)$ be the luminance sample of the current frame at position (i, j), then the four sub-sequences as descriptions are formed with pixels $x(2i,2j)$, $x(2i+1,2j)$, $x(2i,2j+1)$, and $x(2i+1,2j+1)$, respectively. In this way, four sub-sequences with halved resolution on both spatial directions, and a quarter of the original size, correspond to each input sequence [15]. Figure 1 shows an example of coding a 4×4 luma samples into four 2×2 descriptions by SMDC. Since the video source was acquired in planar YUV format of with chroma subsampling 4:2:2, each U and V component whose amount is only half of the amount of Y component needed to be evenly distributed to descriptions by SMDC as well. Figure 2 shows an example of SMDC for evenly coding a 4×4 frame in YUV format with 4:2:2 chroma subsampling into 4 descriptions. As shown in the figure, each description is encoded into 2×2 frame with luminance samples of four Y, chrominance samples of two U and two V.

Figure 3 shows the data flow of preprocessing a video into multiple descriptions. In TuBeck, a video source was acquired in planar YUV format with chroma subsampling 4:2:2. It was preprocessed into four multiple descriptions by SMDC and

* Corresponding author: mikelin@mail.nutn.edu.tw
DOI：10.6159/IJSE.2013.(3-3).03

each was then sent to a separate H.264/AVC encoder to output compressed H.264 videos for efficient video repository and network delivery. We used the JM encoder module, lencod.exe, developed by Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG [20]. Because the restriction of JM encoding process, the width and the height of a video must be a multiple

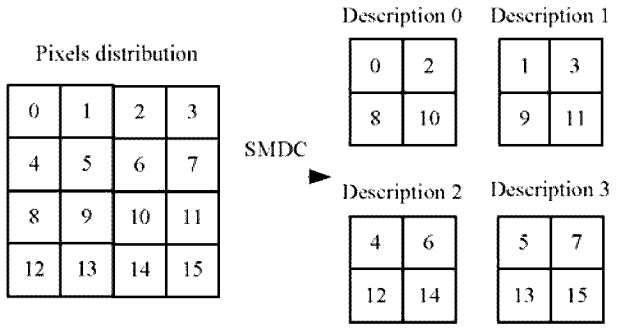


Fig. 1. An example of coding a 4×4 luma samples into 4 descriptions by SMDC.

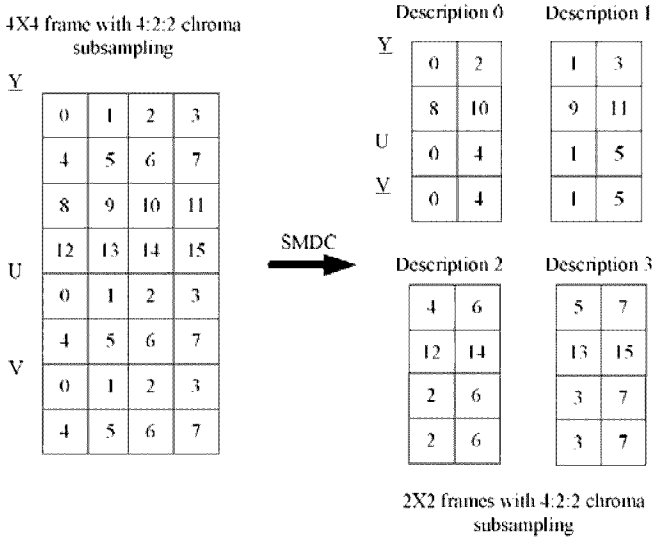


Fig. 2. An example of SMDC for coding a 4×4 frame in YUV format with 4:2:2 chroma subsampling into 4 descriptions.

of 16 and therefore we only adopted CIF (352×288) and QCIF (176×144) resolution videos. In TuBeck, each description was partitioned as a series of chunks which contains 10 frames, one group of pictures (GOP). The frame rate is set to be 30 fps. These important factors were fed into the JM encoder along with the default coding parameters to transcode a description in YUV format into a series of video chunks in H.264 format. These descriptions were stored in hard disks and then accessed for network delivery by requests.

When a peer requests a certain number of descriptions, selected source peers should be distinct to avoid cascade effect on losing viewing quality at child peers when it is failed. Since requested descriptions come from different source peer and travel different network path, they may arbitrarily arrive at the requesting peer. The received chunks of a description are decoded into YUV format and buffered waiting for merging. Intuitively, as long as chunks of various descriptions with the same ID arrive, they should be immediately merged for rendering. However, they probably arrive at the requesting peer at the same time due to network delay or even link broken. The early-coming chunks wait for the late-coming chunks to merge until they are called for playback. Apparently, when to merge chunks of various descriptions with the same chunk ID will affect the viewing quality. On the other hand, since the buffer space is limited, rate control should be applied to avoid buffer overflow and underflow. Based on the above observations, in addition to introducing the system architecture of the proposed player for MDC videos, in this paper we also proposed and studied two control mechanisms, Merging Control Mechanism and Transmitting Rate Control Mechanism to ensure smooth playback without buffer overflow and underflow. Experimental result showed the decoding time of chunks is affected by the number of descriptions, the size of a frame, and the size of a chunk. In addition, the merging time is also affected by size of a frame and size of a chunk. Based on above observation, the size of a chunk equal to 10 frames results in better QoE of users. Furthermore, the proposed transmitting rate control mechanism and merging control mechanism can effectively eliminate the buffer overflow and underflow, which guarantees the smoothness of playback MDC videos. It is noted that our previously proposed TuBeck [19] discuss the system architecture of streaming MDC videos on peer-to-peer networks with the Loopback-MDC, and in this paper we primarily focus on the design and analyses of the performance of the proposed MDC player.

The remainder of this paper is organized as follows. In Section II, we describe the system architecture of the player for MDC videos. In Section III and Section IV, we elaborate the merging control mechanism and transmitting rate control mechanism in detail. In Section V, we present the performance study to evaluate the efficiency of merging control mechanism and transmitting rate control mechanism. Finally, we conclude this paper in Section VI.

II. SYSTEM ARCHITECTURE

The system architecture of H.264 MDC player shown in Fig. 4, is functionally divided into modules which are **JM H.264 Decoder**, **Merger**, **Renderer**, **Transmitting Rate Controller**, and buffer spaces including **Merged Chunks Buffer** and **Decoded Chunks Buffer**. The JM H.264 decoder module is responsible for decoding the incoming chunks from H.264 format to YUV format by utilizing the decoding module of the JM [20]-[22], a H.264/AVC reference software published by the JVT, which was exported as dynamic-link library and linked in our C#.net program. The decoded chunks are then stored in decoded chunks buffer, which is implemented as a hash table for expediting chunk merging process. The merger module is responsible for continuously merging the chunks with the same ID of various descriptions in decoded chunks buffer. The merged chunks are put into merged chunks buffer

which is implemented as a circular queue for efficient memory usage. The timing of merging chunks with the same chunk id depends on the latest merged chunk id and rendered chunk id which are sent by the merged chunks buffer and the renderer, respectively. The renderer continuously fetches chunks from the merged chunks buffer based on the playback rate for rendering. We use threaded timer component to periodically get a frame for rendering in the render module. Since the

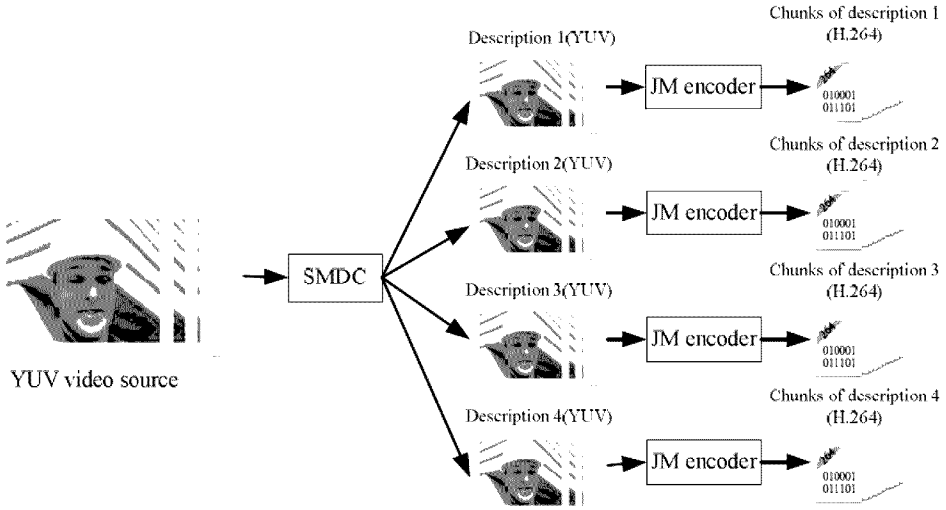


Fig. 3. Data flow of preprocessing a video into multiple descriptions.

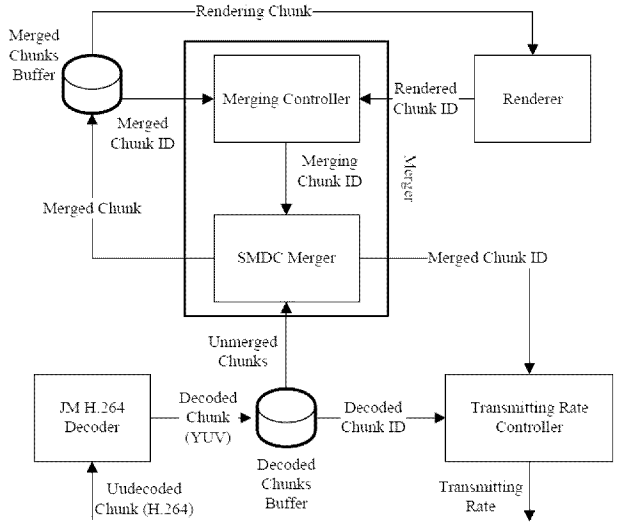


Fig. 4. The system architecture of MDC player.

video is playback at 30 fps, the timer would trigger a signal to get a frame from merged chunks buffer for rendering every $1/30$ second. The transmitting rate controller is responsible for flow control of chunk reception, and the rate adjustment is based on the latest decoded chunk id and merged chunk id which are sent by the decoded chunks buffer and the merger. Based on the above statement, apparently there are two major issues to be addressed to avoid buffer overflow and underflow in order to provide hiccup-free playback. The first one is the timing for merging chunks of description into a video block ready for playback. If the merging of a particular video block is triggered too early, the late-coming chunks of that video block may be left behind and dropped due to network congestions, leading to the reduction of viewing quality. The overflow of merged chunks buffer and the underflow of decoded chunks buffer might also happen. On the other hand, if the merging is too late, the merged chunks buffer might be underflow and the renderer would hunger for video blocks to playback. As a result, the user might experience video freezing. Consequently, it would probably cause the overflow of decoded chunks buffer due to the late merging. Secondly, the transmitting rate of source peers would also affects the status of decoded chunks buffer. If it is too fast/slow, the overflow/underflow of decoded chunks buffer would happen. The availability of chunks in decoded chunks buffer would directly affect the merging quality, it is crucial to control the transmitting rate to match up with the merging speed. In this paper, we propose the Merging Control Mechanism and Transmitting Rate Control Mechanism to address these two issues, which are elaborated in the following Section III and Section IV.

III. MERGING CONTROL MECHANISM

The purpose of merging control mechanism is to determine the timing of merging process in order to timely provide video chunks for smooth playback. In general, the smoothness of playback directly depends on the status of merged chunks buffer. As long as the number of merged chunks is kept within a certain range while playing, the QoE of watching this video can be guaranteed. In this paper, the merged chunks buffer was implemented as circular queue with the capacity of 60 chunks. To efficiently control the merging time, we use two flags. One is Merged Chunk ID, which indicates the latest merged chunks ID. The other is Rendered Chunk ID, which indicates the latest viewed chunks ID by the user. The ”gap” of these two IDs indicates the number of chunks in merged chunks buffer not yet to render. The purpose of the merging control mechanism is to maintain the certain

number of chunks in merged chunks buffer to provide smooth playback. Fig 5 shows an example of a gap of 38 chunks in merged chunks buffer.

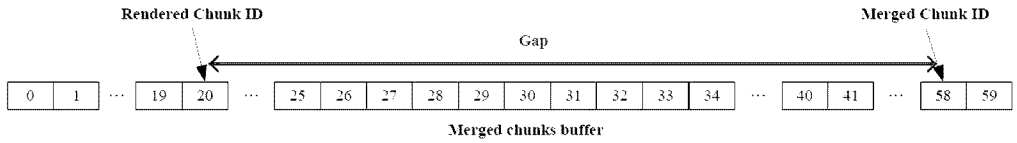


Fig. 5. An example of a gap of 38 chunks in merged chunks buffer.

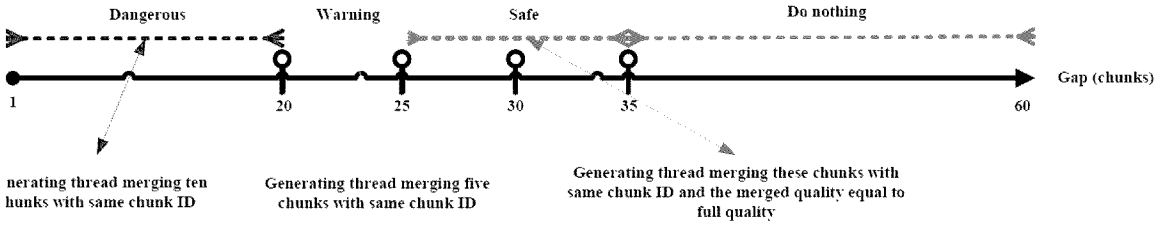


Fig. 6. Merging rate in merged chunks buffer.

Each time the renderer plays a chunk, the merging controller of the merger reads the rendered chunk ID and merged chunk ID from the merged chunks buffer and computes their gap to determine whether the merging process which is executed by the SMDC merger shall be triggered. The value of a gap represents one of the four states of the merged chunks buffer, shown in Fig. 6. First, when the gap is greater than 35, it means the number of chunks in merged chunks buffer is adequate and no merging process is triggered. It is in "do nothing" state. Second, if the gap is within 25 and 35, it is in "safe" state. It means the number of chunks in merged chunks buffer is still adequate but merging these chunks with same chunk ID and the merged quality equal to full quality is necessary now. These chunks are meant to be merged since no chunks with that chunk ID would arrive hereafter. Third, if the gap is within 20 and 25, it means the number of chunks in merged chunks buffer starts to become inadequate and it is in "warning" state. Threads are spawned to concurrently merge 5 chunks to intentionally move the gap back to the "safe" state. It is noted that the merged chunks may not be the one with the full quality. It is our intention in this state to provide partial quality of a chunk if necessary in order to catch up the rendering rate instead of totally missing it. Finally, if the gap is less than 20, it is in "dangerous" state that the small number of chunks in merged chunks buffer would jeopardize the smoothness of the playback. Immediate safety precaution must be taken to avoid buffer underflow. In this state, we spawn threads to merge 10 chunks and try to move the state of the gap back to "warning" state or even "safe" state.

IV. TRANSMITTING RATE CONTROL MECHANISM

The purpose of transmitting rate control mechanism is to avoid the overflow and underflow of decoded chunks buffer due to the mismatch of merging rate and transmitting rate of chunks. When a player receives a chunk, the JM H.264 decoder immediately decodes the received chunks and inserts it into the decoded chunks buffer. There are two flags on decoded chunks buffer, decoded chunk ID and merged chunk ID, which represent the latest decoded chunk inserted into the buffer and the next merging chunk, respectively. In our implementation, the capacity of the decoded chunks buffer is $N \times 20$ decoded chunks, where N denotes the number of descriptions of a video. Each time a decoded chunk is inserted into decoded chunks buffer, the transmitting rate controller estimates the number of chunks in the buffer by computing the difference (gap) of decoded chunk ID and merged chunk ID. Our intention is to control the gap within a safe to avoid buffer overflow and underflow by rate adjustment. To avoid excessive rate adjustment, we classifies the status of the decoded chunks buffer based on the value of its gap into states 0, 1, 2, 3, and 4, which represents the gap within the range of [1, 15], [5, 20], [10, 20], [10, 25], and [15, max], respectively. States 0, 1, 2, 3, and 4 represent the value of a gap of 5, 10, 15, 20, and 25, respectively. As long as the updated gap is still within the current state, there is no rate adjustment occurred. On the other hand, if the updated gap is not within the range of the current state, the state transition occurs and the transmission rate is then adjusted. Figure 7 represents the states of transmission rates. When a state transition occurs, the transmission rate is adjusted based on the following equation,

$$\mathbf{rate_{new}} = \mathbf{rate_{old}} + \frac{\left(\mathbf{gap_{old}} - \mathbf{gap_{now}}\right) \times \mathbf{chunk\_size}}{\mathbf{t_{now}} - \mathbf{t_{old}}},$$

where $\mathbf{t}$ denotes the state transition time. It is noted that the rate is measured by frames per second (fps) and initially, the rate equals to 30 fps. We also set the upper bound and the lower bound of transmission rate which are 90 fps and 10 fps.

V. PERFORMANCE STUDY

We deployed experiments to measure the decoding time and the merging time of various frame size and chunk size with respect to various number of descriptions in a video. The video source haves 100 frames with two types of frame size, CIF (352×288) and 4CIF (704×576) . The SMDC was applied to generate descriptions for a video. Because the JM H.264 encoder only accepts the frame size which must be the multiple of 16, in our experiments, videos with CIF format was encoded as two and four descriptions, and videos with 4CIF format was encoded as two, four, eight, and sixteen

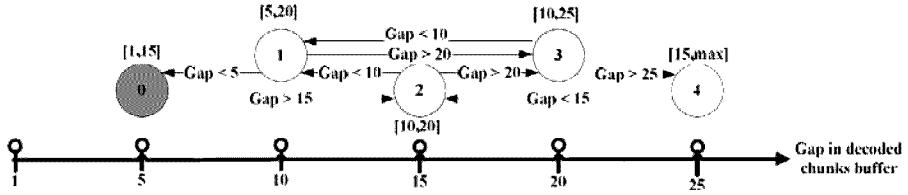


Fig. 7. States of transmission rates.

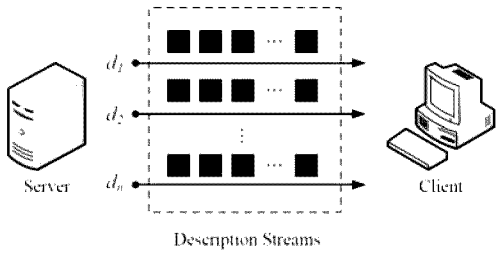


Fig. 8. The emulation environment for the performance study of the MDC Player.

descriptions. Two computers were used to emulate the experiments, one for the source server and the other for the client, which were interconnected in a local-area network. Each requested description was transmitted on a distinct channel with the default transmitting rate of 30 fps, which equals the playback rate. Both equips with Intel Core 2 Quad CPU Q8400 @2.66GHz with 2GB RAM. The experiment lasted 300 seconds. Figure 8 shows the emulation environment for performance study of the MDC player. When a chunk arrives at the MDC player, it must be decoded and merged with other chunks before playback. The overhead of decoding a chunk and merging chunks would definitely affect the quality of the video playback. Figure 9 shows the decoding time of a chunk for different frame sizes and chunk sizes with respect to various numbers of descriptions. The 704×576 (10) denotes the 704×576 video with the chunk size of 10 frames. As shown in the figure, the decoding time of a chunk decreases along with the increase of the number of descriptions and with the same frame size, the decoding time of a chunk increases along with the increase of chunk size. As expected, regardless of the number of descriptions encoded, the chunk of 352×288 (10) has the lowest decoding time and the chunk of 704×576 (30) has the longest decoding time. Note that even the 704×576 (30) video encoded as two descriptions whose rendering time of one chunk equals one second (1000 ms), the decoding time of one chunk is merely 0.36 ms, which is less than 1/1000 of the rendering time. While the same video encoding as 16 descriptions, the decoding time of one chunk is as low as 0.05 ms. Furthermore, it can be concluded that a video is preferably partitioned as a series of chunks with smaller size, such as one GOP with 10 frames shown in the figure, to get a shorter decoding time of one chunk.

Figure 10 shows the merging time of chunks with the same ID for different frame sizes and chunk sizes with respect to various numbers of descriptions. Since the SMDC encodes a video into multiple descriptions in spatial domain, no time-consuming computation is involved during the decoding process. As shown in the figure, in general, the merging time of chunks of the same ID for a video almost remains the same regardless of the number of descriptions encoded. For the same frame size the smaller the chunk size, the less the merging time. For the same chunk size, the smaller the frame size, the less the merging time. It is interesting to see that the merging time for the 704×576 (30) video is only about 0.17 ms. Based on the experiment results of Fig. 9 and Fig. 10, even at the worst case of the 704×576 (30) video encoded as two descriptions, the overheads of decoding a chunk and merging chunks are negligible compared to rendering time of one chunk.

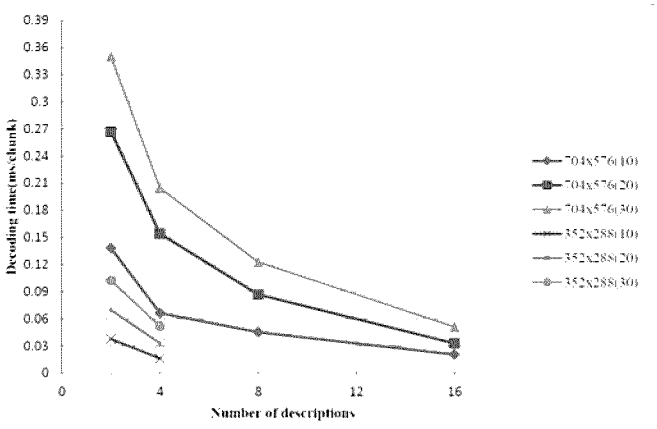


Fig. 9. The decoding time of one chunk for different frame sizes and chunk sizes with respect to various numbers of descriptions.

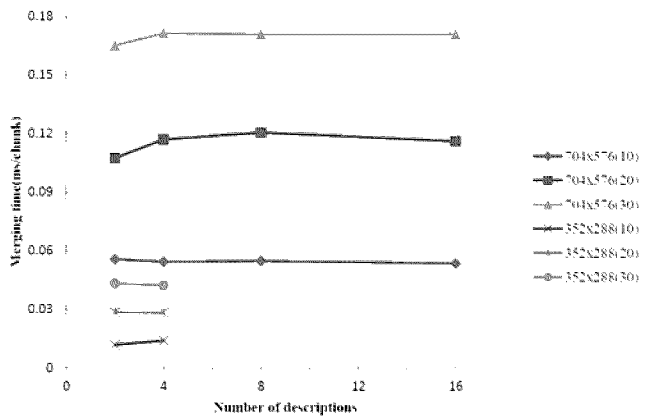
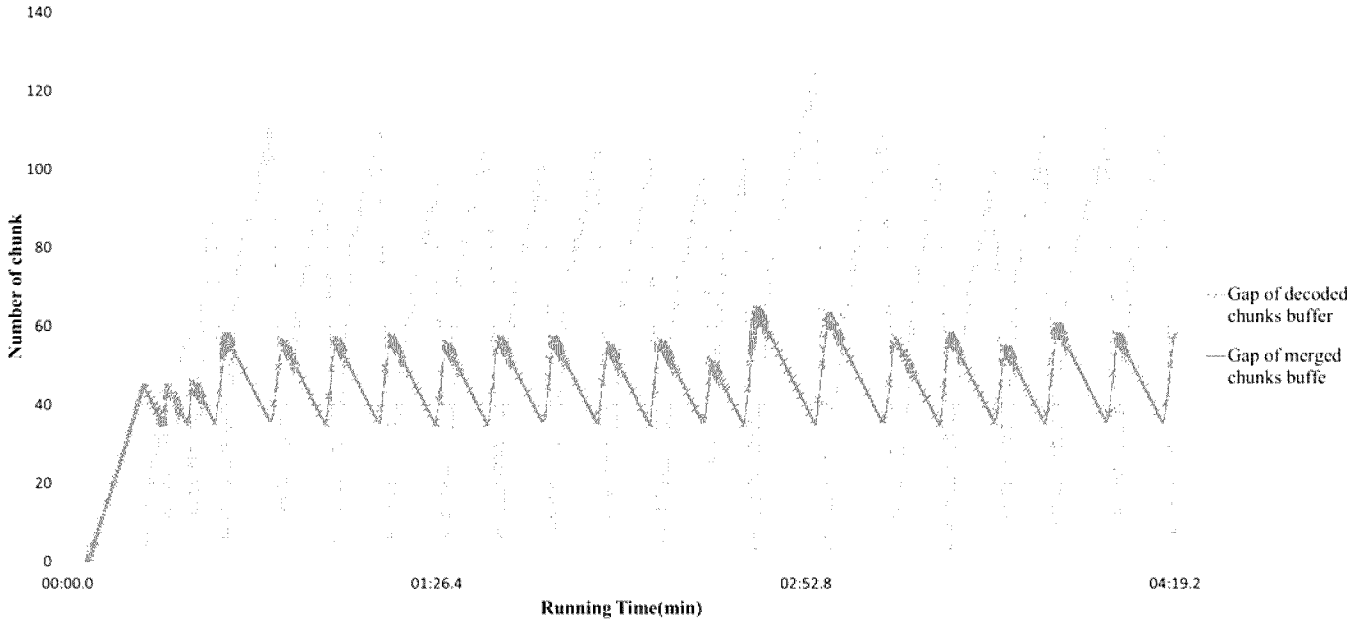
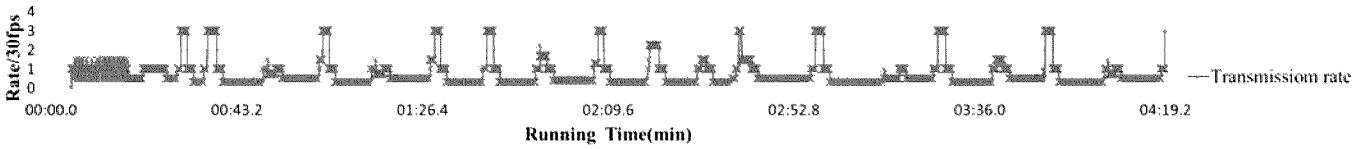


Fig. 10. The merging time of chunks of the same ID for different frame sizes and chunk sizes with respect to various numbers of descriptions.



(a) The gaps of decoded chunks buffer and merged chunks buffer.



(b) The rate adjustment.

Fig. 11. The gaps of decoded chunks buffer and merged chunks buffer, and the corresponding rate adjustment.

Figure 11 shows the gaps of decoded chunks buffer and merged chunks buffer, and the corresponding rate adjustment. Conceptually, the value of a gap represents the number of chunks in decoded/merged chunks buffer. In Fig. 11(a), we can clearly observe the oscillation patterns of decoded and merged chunks buffers. After the initial buffering, the gap of merged chunks buffer decreases along with the decrease of the gap of decoded chunks buffer, and vice versa. The gaps of decoded chunks buffer and merged chunks buffer were nicely controlled within [5, 122] and [62, 38], respectively. No buffer overflow or underflow was shown during the experiments. Figure 11(b) shows the rate adjustment triggered by the transmission rate controller. In the figure the length of each horizontal segment represents the duration of transmission rate without adjustment. It is obvious that the design of states of transmission rates effectively reduces the number of rate adjustment and the rate is well controlled within [10, 90] fps.

VI. CONCLUSION

In this paper, we present the design and implementation of H.264 player for multiple description coded videos. A video is assumed to be encoded into multiple descriptions by the

SMDC. The decoder module of JM is utilized to decoding the received h.264 chunks from the networks. To avoid overflow or underflow in decoded chunks buffer and merged chunks buffer, merging control and transmitting rate control mechanisms are proposed. Experiment results show that with the aids of proposed control mechanisms, no buffer overflow or underflow was experienced during the playback. The overheads of decoding a chunk and merging chunks are negligible compared to rendering time of one chunk, which would not affect the smoothness of playback at all.

VII. REFERENCES

[1] V. K. Goyal, "Multiple description coding: compression meets the network," IEEE Signal Processing Magazine, vol. 18, no. 5, pp. 74– 93, 2001. Doi: 10.1109/79.92/52806

[2] A. Albanese, J. Blomer, J. Edmonds, M. Luby, and M. Sudan, "Priority encoding transmission," IEEE Trans. Information Theory, vol. 42, no. 6, pp. 1737–1744, 1996. Doi: 10.1109/18.556670

[3] Y. Wand, A. R. Reibman, and S. Lin, "Multiple description coding for video delivery," in Proc. of the IEEE, 2005, pp. 57–70. Doi: 10.1109/JRPOC.2004.839618

[4] X. Tan and S. Datta, "Building multicast trees for multimedia streaming in heterogeneous P2P networks," in IEEE Systems Communications, 2005, pp. 141–146. Doi: 10.1109/ICW.2005.33

[5] H. Bai, M. Zhang, M. Liu, A. Wang, and Y. Zhao, "Multiple description video coding using macro block level correlation of inter-/intra-descriptions," in Data Compression Conference (DCC), 2012, april 2012, p. 389. Doi: 10.1109/DCC.2012.45

[6] C.-S. Lin, "Balanced dynamic buffering for scalable video-on-demand streaming on peer-to-peer networks," Multimedia Tools and Applications, published online, Sep. 2011. 10.1007/s11042-011-0872-4

[7] C.-S. Lin and M.-J. Yan, "Service availability for P2P on-demand streaming with dynamic buffering," in IEEE ICPADS, 2011, pp. 741–746. Doi: 10.1109/ICPADS.2011.118

[8] C.-S. Lin, "Enhancing p2p live streaming performance by balancing description distribution and available forwarding bandwidth," **International Journal of Communication Systems**, vol. 23, no. 12, 2010. Doi: 10.1002/dac.1173

[9] C.-S. Lin and I.-T. Lee, "Applying multiple description coding to enhance the streaming scalability on cdn-p2p network," **International Journal of Communication Systems**, vol. 23, no. 5, pp. 553–568, 2010. Doi: 10.1002/dac.1087

[10] C.-S. Lin and W.-T. Syu, "A fine-grained balancing scheme for improved scalability in p2p streaming," **Multimedia Tools and Applications**, vol. 46, no. 1, pp. 71–90, 2010. Doi: 10.1007/s11042-009-0308-6

[11] C.-S. Lin and I.-T. Lee, "Loopback-mdc: Scalability enhancement for streaming multiple description coded video on cdn-p2p network," in BAI, Kitakyushu, Japan, July 2010.

[12] Y. Xu, C. Zhu, W. Zeng, and X. J. Li, "Multiple description coded video streaming in peer-to-peer networks," **Signal Processing: Image** Communication, vol. 27, no. 5, pp. 412 – 429, 2012. Doi: 10.1016/j.image.2012.02.005

[13] V. Padmanabhan, H. Wang, and P. Chou, "Supporting heterogeneity and congestion control in peer-to-peer multicast streaming," in **Peer-to-Peer** Systems III, ser. Lecture Notes in Computer Science, G. Voelker and S. Shenker, Eds. Springer Berlin Heidelberg, 2005, vol. 3279, pp.54–63. Doi: 10.1007/978-3-540-30183-7_6

[14] E. Akyol, A. Tekalp, and M. Civanlar, "A flexible multiple description coding framework for adaptive peer-to-peer video streaming," **Selected Topics in Signal Processing, IEEE Journal of**, vol. 1, no. 2, pp. 231–245, 2007. Doi: 10.1109/JSTSP.2007.901527

[15] O. Campana, A. Cattani, A. D. Giusti, S. Milani, N. Zandona, and G. Calvagno, "Multiple description coding schemes for the h.264/avc coder," in **Proceedings of the International Conference on Wireless Recognizable Terminals and Protocols**, May 2006, pp. 217–221.

[16] O. Campana and S. Milani, "A multiple description coding scheme for the h.264/avc coder," in **Proc. of the International Conf. on Telecommunication and Computer Networks**, 2004, pp. 191–195.

[17] J. Apostolopoulos, "Reliable video communication over lossy packet networks using multiple state encoding and path diversity," in **Proc. of Visual Communications: Image Processing**, 2001, pp. 392–409. Doi: 10.1117/12.411817

[18] N. Zandon'a, S. Milani, and A. D. Giusti, "Motion-compensated multiple description video coding for the h.264/avc standard," in **Proc. of IADAT International Conf. on Multimedia, Image Processing and Computer Vision**, 2005, pp. 290–294.

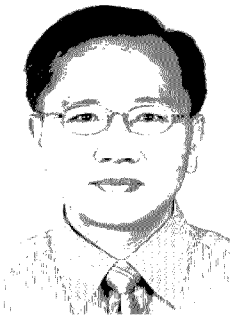
[19] C.-S. Lin, R.-H. Chang, and Jhe-Wei Lin "TuBeck: A Novel Peer-to-Peer Streaming System with Loopback-MDC by Using H.264/AVC Coder," Multimedia Tools and Applications, published on line, DOI: 10.1007/s11042-013-1482-0, April, 2013. Doi: 10.1007/s11042-013-1482-0

[20] Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG, "Joint model (jm) - h.264/avc reference software," ver. 18.0. [Online]. Available: http://iphome.hhi.de/suehring/tml/

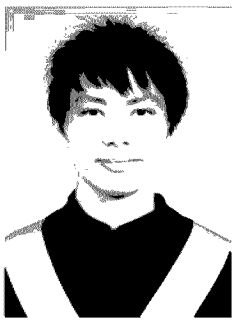
[21] I. Richardson, **H.264 and MPEG-4 Video Compression: Video Coding for Next Generation Multimedia**. Wiley, December 2003.

[22] G. J. Sullivan and T. Wiegand, "Video compression v from concepts to the h.264/avc standard," **Proceedings of the IEEE**, vol. 93, no. 1, pp. 18–31, January 2005. Doi: 10.1109/JPROC.2004.839617

BIOGRAPHIES



Chow-Sing Lin received the Ph.D. degree in Computer Engineering from the University of Central Florida, Florida, USA, in 2000. He is currently a full Professor in the Department of Computer Science and Information Engineering, National University of Tainan, Taiwan. His research interests include wired/wireless media delivery, peer-to-peer media streaming, distributed multimedia systems, mobile computing, and sensor networks. He is listed on the Who's Who in Science and Engineering, 2011-2012 (11th Edition).



Jhe-Wei Lin received his B.S.Eng. degree in Computer Science and Information Engineering from National University of Tainan, Taiwan, in 2011. He is currently a M.S student in the Department of Computer Science and Information Engineering at National University of Tainan, Taiwan. His research focuses on bittorrent with real-time video service, peer-to-peer media streaming, and multiple description coding over P2P network.