IEICE TRANS. COMMUN., VOL.E93–B, NO.3 MARCH 2010

PAPER

Retransmission-Based Distributed Video Streaming with a Channel-Adaptive Packet Scheduler

Young H. JUNG[†a], **Hong-Sik KIM**[††], *Nonmembers, and* **Yoonsik CHOE**[††], *Member*

SUMMARY This paper describes a channel-adaptive packet scheduler for improved error control performance in a peer-cooperative distributed media streaming system. The proposed packet-scheduling algorithm was designed for the case in which streaming server peers rely on an error-recovery strategy using retransmission and application-layer automatic repeat request rather than error protection using forward error correction. The proposed scheduler can maximize retransmission opportunities and reduce the frame loss rate by using the observed channel status from each server peer. Simulation results show that the proposed algorithm enhances error-recovery performance in distributed multimedia streaming better than other schedulers.

key words: *error recovery, retransmission, ARQ, distributed media streaming, peer-to-peer, stored-video streaming*

1. Introduction

Video streaming over the Internet has been a great success in the past decade. In general, video streaming requires a larger bandwidth than other types of Internet services such as Web browsing. This larger bandwidth requirement occasionally causes network congestion and server overload, especially for large-scale services. Distributed multimedia streaming that is based on many-to-one transmission technology has garnered much interest [1], [2] to prevent overloading the streaming server and to use network resources effectively.

In distributed multimedia streaming technology, a client peer receives segments of stream data from multiple server peers using independent connections. Using multiple channels provides fault tolerance, and network resources can be used efficiently because peer-to-peer-like distributed streaming exploits the upload bandwidth of each participating peer, which has never been used in traditional client-server model-based streaming systems. However, the use of these multiple channels and this segmented data stream can also result in additional delay for frame reassembly depending on the transmission order of the segmented data packet and the status of each channel. Additional delay for frame reassembly during streaming usually requires the streaming client to use more pre-roll buffering time or startup delay before playback, and more rebuffering time during playback to avoid starving the playout buffer. Moreover, because this additional frame reassembly delay can also decrease retransmission opportunities for a lost packet when the video streaming service uses error recovery, it can also reduce the objective video quality. Therefore, a packet scheduling scheme must be designed to guarantee minimum delay for frame reassembly both to enhance loss recovery performance and to reduce buffer starvation during distributed multimedia streaming.

To this end, some avenues of research for segment scheduling algorithms have been suggested. Xu et al. proposed a content segmentation and allocation algorithm to guarantee minimum pre-roll delay according to the bandwidth that each server peer can provide [3]. Similarly, Kwon and Yeom suggested fixed-length slotted scheduling exploiting the heterogeneity of the upload contribution among sender peers [4]. Nguyen and Zakhor proposed optimal packet scheduling algorithms that can reduce the probability of packets arriving late at the receiver due to network jitter [5]. Although these studies show the potential of sophisticated packet scheduling in distributed streaming environments to minimize the delay before streaming playback, they did not consider the status of each channel from separate senders. Furthermore, the possibility of network congestion and packet loss were not considered.

Nguyen and Zakhor [6] extended their previous work [5] into a case using forward error correction (FEC) to overcome transmission errors in each distributed channel. They also considered redundant packets for FEC in the case of a packet scheduling procedure. They minimized the probability of unrecoverable packet loss and guaranteed minimum frame reassembly delay using this scheduling scheme. However, in their algorithm, they assumed the availability of sufficient bandwidth from each streaming server, as well as prior knowledge of the channel information such as the packet loss rate. In general, although FEC imposes no additional delay, it does have the drawbacks of requiring additional bandwidth and prior knowledge of the channel status. Moreover, because the bitstream with fixed FEC is assumed to be allocated to multiple senders, it will be less efficient if it is used by multiple senders in a heterogeneous and dynamically varying channel environment. Therefore, especially in a wireless environment that is less predictable and that generally has insufficient bandwidth for each user, FEC is less efficient than automatic repeat request (ARQ). In particular, in a peer-to-peer (P2P) service like distributed streaming, it is more beneficial for each server peer to reduce its upload bandwidth contribution by using ARQ rather than FEC [7].

Manuscript received March 13, 2009.
Manuscript revised October 24, 2009.
[†]The author is with Samsung Electronics. Co. Ltd., Suwon, Korea.
[††]The authors are with School of Electrical & Electronic Engineering, Yonsei University, Seoul, Korea.
a) E-mail: crosscom@yonsei.ac.kr
DOI: 10.1587/transcom.E93.B.696

This paper proposes a channel-adaptive packet scheduling algorithm designed to provide minimum delay in frame reassembly and to enhance the ARQ-like error-recovery performance in a distributed video streaming environment. Although normal ARQ is not suitable for real-time streaming applications, an enhanced version of ARQ called delay-constrained ARQ or SoftARQ [8] works well because it is based on a time-constrained selective ARQ principle. Therefore, the objective of our work is to design a packet scheduler based on the channel status and ARQ-like behavior for better streaming quality in a distributed streaming system. To achieve this, the client peer first observes the channel status of multiple server peers. The scheduler in the client peer determines the candidate server peer that is to send each packet of the media stream according to the expected packet arrival time of each packet. The expected packet arrival time from each server is calculated based on the upload bandwidth of the server peer and the observed current channel status. After selecting the server peer for each packet that yields the minimum expected packet arrival time of all the candidate server peers, the packet scheduling map for one group of pictures (GoP) is generated in the distributed streaming client application. The delay in reassembling the frame can be minimized using this proposed packet scheduling algorithm even in channel errors. Moreover, the reduced delay in frame reassembly helps to provide more retransmission opportunities for subsequent lost packets. Eventually, the frame recovery performance can be improved and overall frame distortion can be minimized by the proposed scheduling algorithm.

The major contribution of this paper is to demonstrate that error-recovery performance can be enhanced using a sophisticated packet scheduler in a distributed stored-video streaming system. Simulation results show that the proposed scheduler can enhance the quality of service (QoS) of streaming by reducing the probability of both packet and frame loss. Therefore, the proposed algorithm is efficient in applications such as mobile P2P streaming services that have limited server peer upload bandwidth and dynamically changing channel status.

The rest of the paper is organized as follows. We define the service system architecture and model with some essential assumptions in Sect. 2. In Sect. 3, we present the packet-scheduling problem and the error-recovery process. In Sect. 4, we describe the role of the client and server in the proposed system, and in Sect. 5, we explain the proposed channel-adaptive scheduler in detail. We present the simulation results in Sect. 6 and conclude in Sect. 7 with a discussion of areas for future study.

2. Network Model and Service Architecture

Here we present the service and system architecture considered in this paper along with the assumptions we made. In this work, we supposed peer-cooperative distributed streaming service which has the feature that each peer should act as both server and client for the service activation. For this distributed streaming service, following assumptions are introduced:

- Each peer caches certain amount of video data chunks for uploading to others.
- Several peers who have been watching specific content co-exist in the network.

Note that, in this research, we excluded other research topics except for packet scheduling, such as finding appropriate sending peer or optimally caching video data in distributed peers. We just focused on packet scheduling algorithm for in-time packet delivery to enhance streaming quality as a building-block of successful peer-cooperative distributed streaming service.

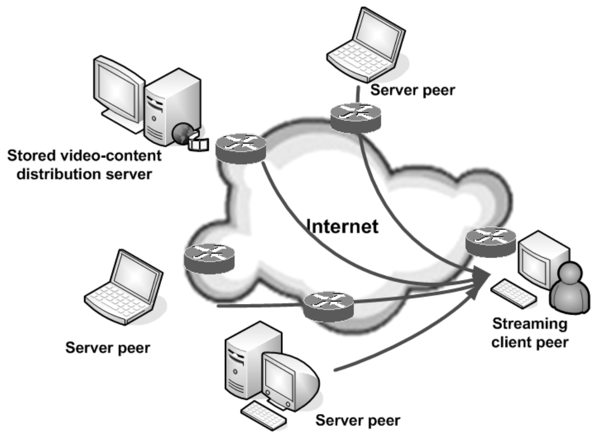

Fig. 1 Service architecture for distributed stored-media streaming.

As described in previous works [3], [5]–[7] and shown in Fig. 1, our target service is also on-demand video streaming from a pre-encoded video source. In general, this stored-video streaming service uses pre-roll buffering time or startup buffering and has fewer strict time constraints than live video streaming. Because the video content is already encoded and stored on server peers, the streaming client can first obtain and exploit playback information such as the number of packets per frame and the total number of frames. For this distributed stored-video streaming service, we assume that server peers that have requested the content are distributed over the network and they have heterogeneous upload bandwidth to serve streaming service for other peers. From the transport layer perspective, User Datagram Protocol (UDP) and Real-time Transport Protocol (RTP) are used with simple feedback control packets. From the perspective of the system architecture, we assume that the client has the responsibility for the packet scheduling algorithm. That is, the client makes a packet sending sequence called a scheduling map for each sender, and the client delivers this scheduling map to each sender using the feedback control channel prior to playing each GoP.

Delay-constrained ARQ [8], [9] is used in this system for error recovery of the video stream data. This includes playout buffering, gap-based loss detection, and conditional retransmission schemes. Playout buffering involves prior pre-roll buffering to display the received video stream data; this process is already widely used in commercial media-streaming applications. Gap-based loss detection means that packet-loss detection is based on the gap between two successive transmitted packets; it is not determined by a timer and acknowledgment packet (ACK) for each transmitted packet. Furthermore, conditional retransmission means that retransmission of a lost packet is intentionally skipped if the lifetime of the packet has already expired. In addition, we assume that retransmission packets have a higher priority than normal scheduled packets. This simplifies the evaluation and implementation.

Since appropriate peer selection is another research area, we assume that sufficient numbers of appropriate server peers that have a unique sever identity (S_ID) have already been chosen. Moreover, we assume that the streaming video profile containing the frame size and GoP structure has already been communicated to the client. As in [7], we also assumed that each sender sends packets at a constant rate and denote the time needed to send one packet as one time unit. Furthermore we assumed that the state transition in the channel occurs at packet level discrete time unit and that the steady-state packet loss probability is almost constant. Therefore, each channel could be assumed to be a Gilbert-Elliot channel, and it was also assumed that network conditions for the channels of each sender are independent.

3. Frame Reassembling Delay during the Retransmission

Nguyen and Zakhor suggested a packet partitioning algorithm (PPA) to minimize startup delay [6]. The PPA can also result in a minimum delay for frame reassembly because the reassembly delay of the initial frame is the same as the startup delay in that paper. We use the term "minimum delay for frame reassembly" rather than "minimum startup delay" as we assume a fixed playout buffering delay value.

The PPA is executed at individual servers using information from the client with a relatively simple equation for the expected packet arrival time. That is, sender i that has the minimum expected packet arrival time $A_i(n)$ in Eq. (1) should be chosen to send the nth packet of the streaming data.

$$A_i(n) = n_i \cdot T_i(1) + 2D_i \tag{1}$$

where n_i is the number of packets already sent by sender i up to packet n, $T_i(1)$ is the unit time to send one packet for sender i, and D_i is the estimated delay from sender i to the receiver. This is because it takes D_i for the scheduling map packet to arrive at the sender i through the feedback control channel, $n_i \cdot T_i$ for the nth packet to be sent by sender i, and D_i for it to arrive at the receiver.

This PPA can optimally schedule and allocate each packet to the appropriate distributed sender while minimizing the frame reassembly delay because it leads to the minimum packet arrival time for each packet. However, if this

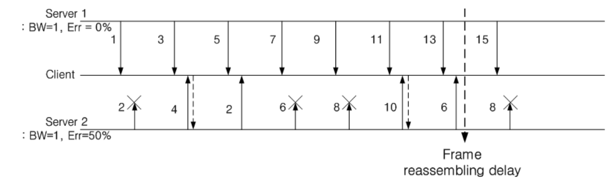

Fig. 2 Delay for frame reassembly with PPA scheduler.

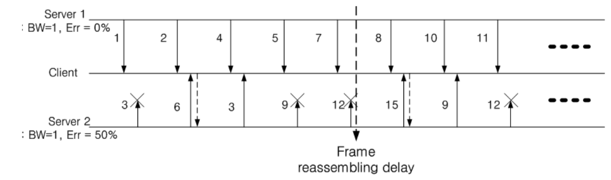

Fig. 3 Desired delay for frame reassembly.

scheduling is used in conjunction with ARQ, then it will not always produce the optimal scheduling solution. For example, let us assume two senders that have equal sending bandwidth (BW = 1) but quite different instantaneous loss rates (0% versus 50%). With these senders, if one frame is composed of seven packets (sequence ID 1-7), then the delay for frame reassembly in the case of the PPA can be seven time slots as shown in Fig. 2. If, however, we use a packet schedule like that shown in Fig. 3 in the same environment, the delay for frame reassembly can be reduced to five time slots. Our objective is to find a near-optimal packet-scheduling algorithm for a retransmission-based system that will provide the minimum delay for frame reassembly. This reduced delay for reassembling frames can also provide more opportunities for the retransmission of subsequent lost packets and may result in a lower video frame loss rate.

4. System Design

This section describes the role of the client and server in the proposed retransmission-based streaming system. In the system design, the functionality of packet retransmission and packet scheduling is allocated to network adaptation layer (NAL) application to avoid OS kernel change. This NAL application is located in the middle of core streaming application and transport layer. In this paper, we just denoted this NAL as an "application."

The server peer application in this system has these relatively simple functions:

- Store the scheduling map from the client peer.
- Send the stream packets assigned to it in the scheduling map.
- Send retransmission packets with the highest priority whenever it receives an ARQ from the client peer.

The client peer application in this system has more complex functions than the server:

- Generate the scheduling map according to the proposed algorithm.
- Observe the channel status parameters such as average

loss rate or loss model transition probability.

- Check the sequence gap using received and expected packet identity (P_ID).
- Send the ARQ message to the server to which the lost packet is allocated.
- Check the playout buffer (PoB) status and attempt re-buffering when the PoB level falls below the predefined threshold.

We use two different identities for each packet during streaming to check for packet loss based on the sequence gap and channel status. The P_ID is used among whole server peers, and therefore, each packet from server peers contains a unique P_ID. If a retransmission takes place, the client sends an ARQ with this P_ID and the server peer also sends the packet with the requested P_ID. Another identity, the local sequence identity (LSEQ_ID), is used locally by each server peer. This LSEQ_ID is incremented by one each time the server peer sends a packet, for both normal and retransmission packets. The client peer can calculate the channel status from each server peer using this LSEQ_ID. That is, the gap of the LSEQ_ID between two successive received packets means the packet loss in the transmission channel. By using this identity, the client peer counts the number of good-to-bad and bad-to-good state transition event, and updates the state transition probabilities, which will be used for the scheduling algorithm.

5. Channel-Adaptive Packet Scheduler

To reflect the channel status during packet scheduling, the packet scheduler first continuously observes the channel status of each distributed channel. As mentioned in Sect. 2, we modeled the packet erasure channel as a two-state Markov channel, which is called the Gilbert-Elliot channel [10]. That is, each channel c_i for sender i has the good-to-bad and bad-to-good transition probability, p_i and q_i, respectively, as shown in Fig. 4.

From the steady-state analysis, the average bad status rate, L_i, can be calculated as

$$L_i = \frac{p_i}{p_i + q_i} \tag{2}$$

Let S_i^0 be the status of channel c_i for the packet under scheduling consideration. If this packet is lost (i.e., assigned "bad" status), at least one other packet must be transmitted successfully for gap-based loss detection. After the detection of the packet loss by the sequence gap, the retransmission packet should be scheduled to be sent by the server because it is assumed that the ARQ packet is delivered immediately from the client to the server. Let $G_i(k)$ be the probability that the client receives a packet successfully after k successive packet losses (Fig. 5). Then, $G_i(k)$ can be represented with the conditional probability.

$$\begin{aligned} G_i(k) &= p[S_i^k = G, S_i^{k-1} = B, \cdots, S_i^1 = B, S_i^0 = B] \\ &= L_i q_i (1-q_i)^{k-1},\ k \geq 1 \end{aligned} \tag{3}$$

where S_i^k is the status of channel c_i after the kth time-slot ($k \geq 1$). Furthermore, G and B in Eq. (3) means transmission success state (Good) and fail state (Bad), respectively. Then, we can calculate the probability $R_i(k)$ that the first lost packet will be successfully retransmitted after sending k time-slots from the trellis of channel status by

$$\begin{aligned} R_i(k) &= G_i(k-1) \times (1-p_i) \\ &= L_i q_i (1-q_i)^{k-2} \times (1-p_i),\ k \geq 2 \end{aligned} \tag{4}$$

The possible packet arrival time can be calculated from the probability and the required time for each of two cases: the case of successful transmission without retransmission and the case of retransmission after several packet losses. Therefore, the possible packet arrival time can be approximated as

$$\begin{aligned} A_i \approx n_i \Bigg[&(1-L_i)(T_i(1) + 2D_i) \\ &+ \sum_{k=2}^{K} R_i(k)(T_i(k+1) + 2D_i) \Bigg] \end{aligned} \tag{5}$$

where K is the number of possible retransmission opportunities. Using this equation, we can reflect ARQ behavior in the scheduling cost function; retransmission probability and its expected time are applied to the estimated successful packet reception time. Because we deployed delay-constrained ARQ, the summation in Eq. (5) should be performed over the period when the packet is still valid. Therefore, K can be determined by the term of the estimated packet lifetime. However, since the packet scheduler cannot accurately anticipate the lifetime of a packet to be sent, the lifetime must be estimated during the scheduling process. In our implementation, the lifetime of each packet is simply estimated as

$$LT\ (\text{sec}) = \frac{F_{PoB} - F_{curr}}{1/fr} \tag{6}$$

where LT means the life-time of the packet under consideration for scheduling, F_{PoB} is the most recently received highest frame number, F_{curr} is the most recently played frame

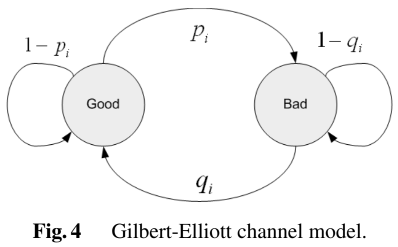

Fig. 4 Gilbert-Elliott channel model.

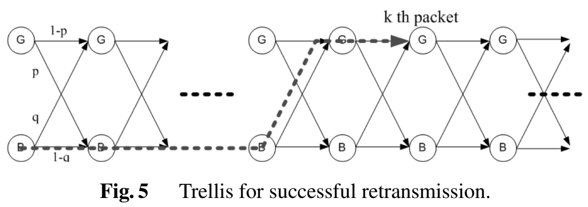

Fig. 5 Trellis for successful retransmission.

number at the client, and fr is the frame rate of the video stream. With this LT, we can determine K in Eq. (5) to satisfy the following condition:

$$\arg \max_{2 \le k \le R_{max}} (k(T_i(1) + 2D_i) \le LT) \tag{7}$$

where R_{max} means the service defined maximum retransmission number. Like the PPA [6], the sender i that has the minimum expected packet arrival time $A_i(n)$ for packet n is selected in the scheduling map. In our implementation of the system, the packet scheduling map decision is performed for every GoP during the streaming process to reflect the updated channel status.

6. Simulation Results

We used network simulator, NS-2 [11] to demonstrate the proposed packet scheduling algorithm and distributed streaming system. Commercial MPEG-4 Simple Profile video streams were used for the simulation. We grouped one I-VOP and afterward successive P-VOPs as one GoP unit. In addition, these video clips were generated to have the frame rate of 30 fps. We used the case of two or three senders with heterogeneous upload bandwidths and heterogeneous channel characteristics. In this simulation, BW_i refers to the contributing upload bandwidth of server i. Furthermore, we assumed that the average packet loss probability of the channel is constant which is denoted by CHi. We used the frame loss rate as a key QoS measure to evaluate the performance of the proposed system. We simulated three different systems for the performance comparison: the proposed system with a channel-adaptive packet scheduler, the system with the PPA [6] scheduler, and the system with a round-robin packet scheduler.

6.1 Static Channel Error Case

To evaluate the performance of error recovery for the heterogeneity of contributing bandwidth among server peers, we first simulated a simple two-sender case. This simple topology was designed simply to examine the frame reassembly delay and frame recovery performance of each scheduler according to the heterogeneity of the upload bandwidth and various sender channel error cases. For this case, we used a test video sequence that had average and peak bitrates of 596 and 1007 kbps, respectively. The channel characteristics and contributing bandwidth of the two senders were set to the values shown in Table 1.

Figure 6 and Table 2 the result of each scheduler in one representative case in which server peer1 has an upload bandwidth of 500 kbps and a channel with an average loss probability of 0.08 while server peer2 has an upload bandwidth of 400 kbps and a channel with an average loss probability of 0.02. In Table 2, AVG_D, DEV_D and MAX_D mean the average, standard deviation and maximum delay for frame reassembly. In addition, FR_LOSS means average frame loss ratio as a quality measure. Moreover, RR, PPA, and $PROP$ mean three comparative systems, round-robin scheduler, PPA scheduler and the proposed scheduler, respectively. Figure 6 is a histogram of the delay for frame reassembly measured using the difference between two packets of each frame: the first received packet of the frame and the last received packet of the complete frame. As shown in Fig. 6 and Table 2, the proposed scheduler has a lower average frame reassembly delay and a smaller standard deviation of frame reassembly delay than the other schedulers. This reduced frame reassembly delay contributes to increasing retransmission opportunities in this retransmission-based streaming system and therefore, the proposed scheduler enhances error recovery performance as shown in Table 2.

In Figs. 7–9, the x-axis or K indicates the proportion of the contributing bandwidth of two senders (BW1/BW2). As the heterogeneity of the contributing bandwidth between the two senders increases, the round-robin scheduler shows severe degradation of video streaming quality. However, the PPA and the proposed scheduler are affected less by the bandwidth heterogeneity of the server peer. Overall, the frame loss rate of PPA varied from 2.067% to 4.170%, whereas that of the proposed scheduler varied from 1.069% to 1.866%. This means that the proposed scheduler is more error-resilient and is affected less by the contributing band-

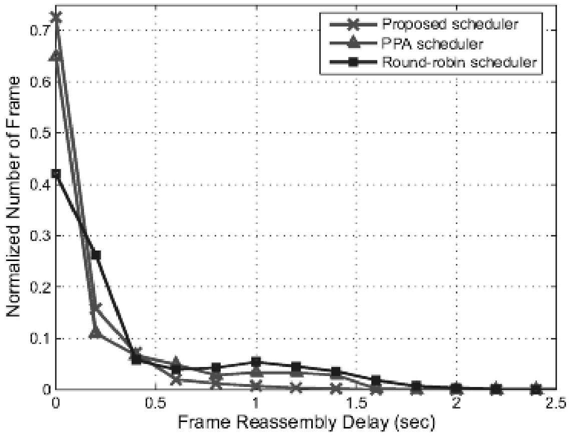

Fig. 6 Histogram of the delay for frame reassembly.

Table 1 Channel parameters.

	p_i	q_i	L_i	UP_BW
CH1	0.98–0.90	0.02–0.10	0.02–0.10	450 kbps–630 kbps
CH2	0.98	0.02	0.02	450 kbps–270 kbps

Table 2 Simulation result. (BW1=500 kbps, BW2=400 kbps)

	AVG_D (sec)	DEV_D (sec)	MAX_D (sec)	FR_LOSS (%)
RR	0.47827	0.62991	5.252764	17.5786
PPA	0.291652	0.397973	1.966457	3.9153
PROP	0.167424	0.217073	1.854653	0.7098

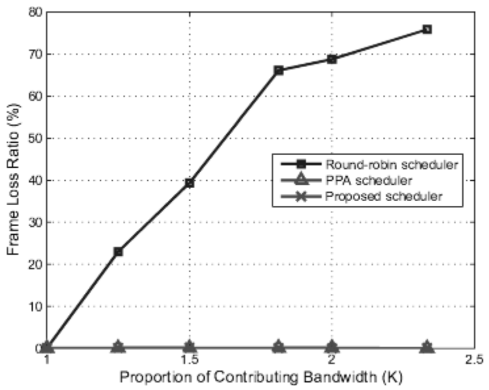

Fig. 7 Frame loss ratio variations for bandwidth heterogeneity. (average loss rate of CH1 = 0.04)

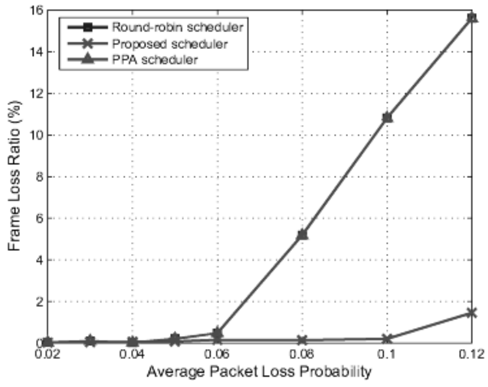

Fig. 10 Frame loss ratio over the variation of channel 1. (450 kbps–450 kbps case)

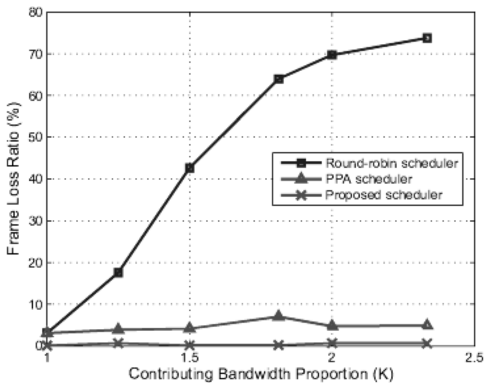

Fig. 8 Frame loss ratio variations for bandwidth heterogeneity. (average loss rate of CH1 = 0.08)

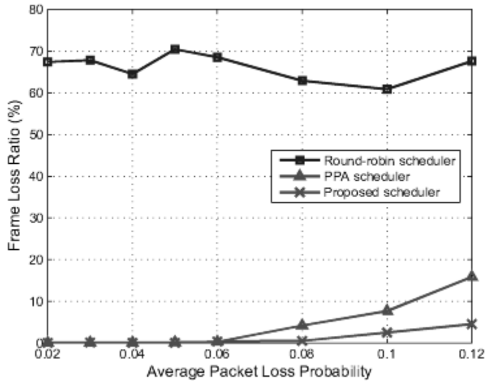

Fig. 11 Frame loss ratio over the variation of channel 1. (550 kbps–350 kbps case)

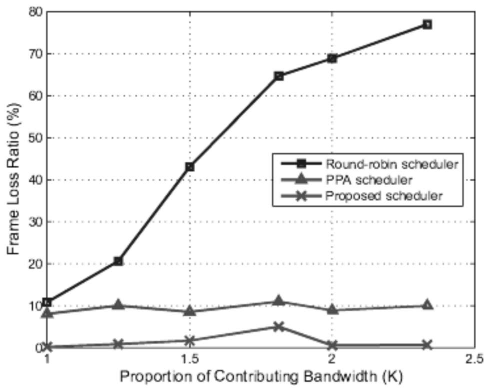

Fig. 9 Frame loss ratio variations for bandwidth heterogeneity. (average loss rate of CH1 = 0.10)

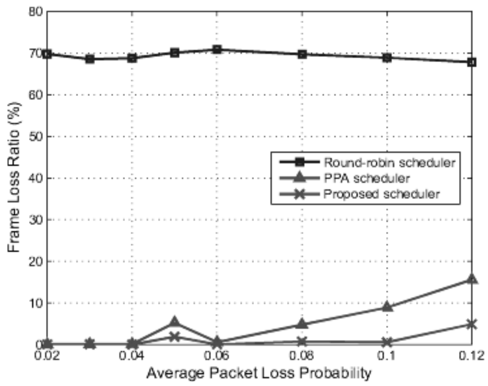

Fig. 12 Frame loss ratio over the variation of channel 1. (600 kbps–300 kbps case)

width heterogeneity of the server peer.

Various channel statuses can also produce an effect on the performance of error recovery as described in Sect. 3. To demonstrate this, we designed another simulation scenario with the environment described in Table 1. That is, upload bandwidth contribution, BW1 and BW2 are fixed in each case and the average loss probability of CH1 varied from 2% to 12% to model the case of a relatively error-prone channel. Figures 10–12 shows a comparison of the frame loss rate for the various simulation cases. The contributing upload bandwidth of server1 increases from 450 kbps (Fig. 10) to 600 kbps (Fig. 12). When the upload bandwidths of the three server peers are the same as shown in Fig. 10 (i.e., the upload bandwidth of the three server peers is homogeneous), the round-robin scheduler and the PPA exhibit the same error-recovery performance under various channel error scenarios. That is because the PPA scheduler allocates packets to each sender in a manner similar to the round-robin sched-

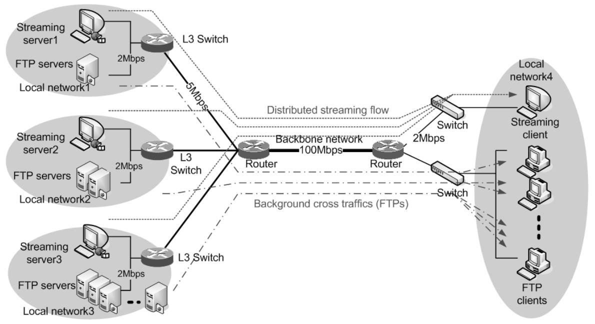

Fig. 13 Simulation topology for congestion over wired network.

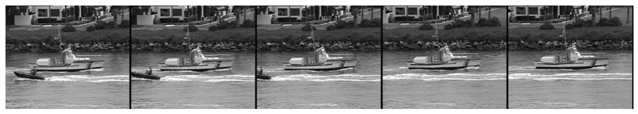

Fig. 14 Original images. (90th, 700th, 710th, 720th, and 730th frames)

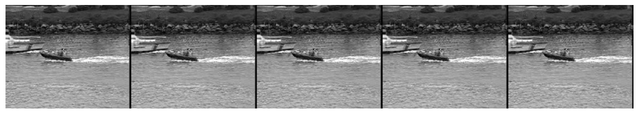

Fig. 15 Decoded images in the system with round-robin scheduler.

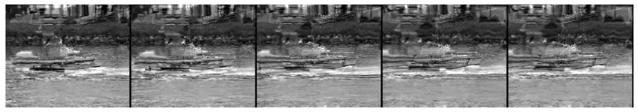

Fig. 16 Decoded images in the system with PPA scheduler.

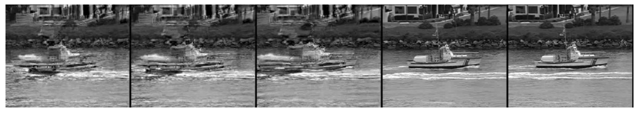

Fig. 17 Decoded images in the system with the proposed scheduler.

uler as the upload bandwidths of each server peer are closer to the same value. Therefore, in cases of homogeneous upload bandwidth and high error rates (Fig. 12), the two schedulers experience similar frame loss rates. However, in the other cases when the upload bandwidth of the three peers is heterogeneous, the proposed scheduler and the PPA significantly outperform the simple round-robin scheduler. This simulation result shows that the error recovery performance of conventional schedulers for distributed media streaming systems depends heavily on both bandwidth heterogeneity and the channel error rate of each server peer, whereas the proposed scheduler shows consistently good error recovery performance under various heterogeneous bandwidth and channel error scenarios. Overall, the proposed scheduling algorithm outperforms the PPA by 3.299% on average in the simulation cases of greatest channel error.

6.2 Simulation of Congestion over a Wired Network

We created the realistic simulation scenario described in this subsection to test the proposed scheduling algorithm. As shown in Fig. 13, streaming servers for a distributed streaming session were scattered over a backbone network and local networks away from the streaming client. Cross-traffic was generated using FTP sessions to create network congestion and ultimately cause queue-drop in the L3 switches of the local networks. The upload bandwidths of server1, server2, and server3 were set to 200, 250, and 350 kbps, respectively. Several cross-traffic flows in each local network were generated: one in local network1, two in local network2, and three to six in local network3. To emulate random packet drop in a real-world network, we configured a random-early-drop (RED) queue in the L3 switch for the outgoing interface to the backbone router. RED parameters were configured as a queue length of 250, drop probability of 0.01 at the low threshold (20% of queue length), and drop probability of 0.9 at the high threshold (80% of queue length). The streaming session was started randomly within a period of 10 s, and the simulation was repeated 100 times.

Figures 14–17 show the simulation results for the case of router queue-drop events when the four cross-traffic flows were generated in local network3. The average peak signal-to-noise ratio (PSNR) of the proposed scheduler is higher than that of the PPA scheduler by 0.52 dB. This relatively small difference of PSNR between the PPA and the proposed scheduler results from error concealment scheme which was applied into the MPEG-4 decoder in these simulations. However, we can verify that the subjective image quality of the proposed scheduler is much better than that of the PPA scheduler as shown in figures. Figure 18 shows the average rebuffering occurrence of each scheduler for various cross-traffic flows in local network3. The frame loss rate of the round-robin scheduler was omitted from this figure because it averaged 47.8%, much larger than the others. As the background FTP flows increased, the streaming systems using each scheduler suffered more video frame loss. However, the proposed scheduler always showed less frame loss than others on average. Therefore, we confirmed that the proposed algorithm would be useful in the case of wired network congestion that causes queue-drop in the router with RED queues. In addition, we configured a drop-tail queue in the L3 switch of each local network. The drop-tail queue size was limited to 150 packets and the same simulation methodology was applied. In this experiment, the proposed scheduler could not always guarantee better streaming quality during network congestion. This is because a drop-tail queue does not use intentional dropping with a certain probability and therefore packet losses are temporarily correlated [12]. Since the proposed algorithm performs channel estimation within the GoP time interval (2 s in this simulation), it can often fail to estimate the temporarily correlated channel status. Therefore, a shorter GoP interval would be used for more precise estimation according to the queue-length of a drop-tail queue in an actual network. However, its application in real-world situations is not trivial.

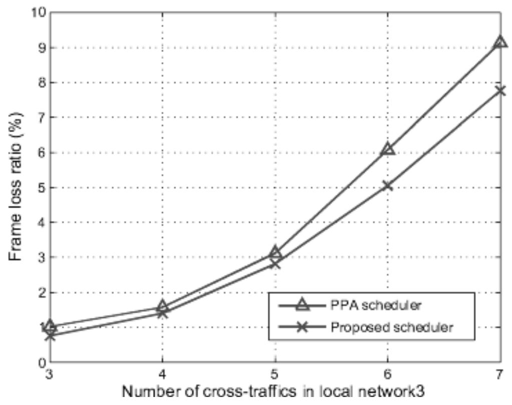

Fig. 18 Frame loss ratio under variation of cross-traffic flows in local network3.

7. Conclusions

We proposed a channel-adaptive packet scheduler to enhance the error-recovery performance by minimizing the delay for frame reassembly in distributed streaming. The proposed scheduler observes distributed channels with the state transition probability of the Gilbert-Elliot channel. Using this observation, each stream packet is allocated to servers to minimize the expected packet arrival time. We demonstrated the effect of the packet scheduler on error-recovery performance in a distributed streaming system using simulations of various cases. Note that the proposed system shows enhanced quality of streaming service, while the receiver in the system should compute entire allocation of the packet stream comparing to the original PPA. However, this computational overhead was approximated enough to be performed in real-time environment. Our future research will be directed toward a more complex and complete architecture of this retransmission-based P2P streaming system including a packet scheduler, soft ARQ mechanism, and ARQ target server derivation. Furthermore, since it can also deliver better streaming quality to reflect the importance of video data in each packet, we will investigate a combinatorial framework that can deal with all of these considerations.

References

[1] J. Liu, S. Rao, B. Li, and H. Zhang, "Opportunities and challenges of peer-to-peer internet video broadcast," Proc. IEEE, vol.96, no.1, pp.11–24, Jan. 2008.

[2] D. Jurca, J. Chakareski, J.-P. Wagner, and P. Frossard, "Enabling adaptive video streaming in p2p systems [peer-to-peer multimedia streaming]," IEEE Commun. Mag., vol.45, no.6, pp.108–114, June 2007.

[3] D. Xu, M. Hefeeda, S. Hambrusch, and B. Bhargava, "On peer-to-peer media streaming," Proc. 22nd International Conference on Distributed Computing Systems, 2002., pp.363–371, 2002.

[4] J.B. Kwon and H.Y. Yeom, "Distributed multimedia streaming over peer-to-peer network," Euro-Par 2003, 9th International Conference on Parallel and Distributed Computing, 2003.

[5] T. Nguyen and A. Zakhor, "Distributed video streaming over internet," SPIE Multimedia Computing and Networking, 2002.

[6] T. Nguyen and A. Zakhor, "Multiple sender distributed video streaming," IEEE Trans. Multimedia, vol.6, no.2, pp.315–326, April 2004.

[7] L. Ma and W.T. Ooi, "Retransmission in distributed media streaming," NOSSDAV'05: Proc. International Workshop on Network and Operating Systems Support for Digital Audio and Video, pp.117–122, New York, NY, USA, 2005.

[8] M.G. Podolsky, S. McCanne, and M. Vetterli, "Soft arq for layered streaming media," J. VLSI Signal Process. Syst., vol.27, no.1-2, pp.81–97, 2001.

[9] C. Papadopoulos and G.M. Parulkar, "Retransmission-based error control for continuous media applications," Proc. 6th Intl. Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV), pp.5–12, 1996.

[10] M. Mushkin and I. Bar-David, "Capacity and coding for the gilbert-elliot channels," IEEE Trans. Inf. Theory, vol.35, no.6, pp.1277–1290, Nov. 1989.

[11] ns-2 Network Simulator, available from http://nsnam.isi.edu/nsnam/index.php/main_page, 2007.

[12] W. Singh, H. Yousefi'zadeh, and H. Jafarkhani, "A finite-state markov chain model for statistical loss across a red queue," Proc. Systems Communications, pp.305–310, Aug. 2005.

Young H. Jung received the B.S., M.S., and Ph.D. degrees in Electrical and Electronic Engineering from the Yonsei University, Seoul Korea in 1997, 1999, 2009 respectively. Since 2000 he has been with Telecommunication division of Samsung Electronics, Co. Ltd., Suwon, Korea, where he is a senior research engineer, focusing on the development of 3GPP/LTE packet network elements and services. His current research interests include multimedia communication over the wired/wireless network, and the multimedia service architectures in 3GPP and 3GPP LTE.

Hong-Sik Kim received the B.S., M.S., and Ph.D. degrees in Electrical and Electronic Engineering from Yonsei University, in 1997, 1999, 2004, respectively. He was a Post-Doctorial Fellow at Virginia Tech, in 2005, VA, and a Senior Engineer at System LSI Group in Samsung Electronics Co, in 2006. He is currently a research professor at Yonsei University, Seoul, Korea. His current research interest includes design for testability, built in self test, and test compression algorithm.

Yoonsik Choe received the B.S. degree in electrical engineering from the Yonsei University, Seoul Korea in 1979, and the M.S.E.E. in systems engineering, M.S. and Ph.D. degrees all in electrical engineering from the Case Western Reserve University, the Pennsylvania State University, and the Purdue University, in 1984, 1987, and 1990, respectively. From 1990 to 1993, he was a principal engineer at the Hyundai Electronics Industries, Co. Ltd. Since 1993 he has been with the school of Electrical and Electronic Engineering at the Yonsei University, where he is a Professor. His research interests include video coding, video communication, and digital image processing.