A Dynamic Rate Switching Method for Live Video Streaming Using a Smart Device

Natsuki Kai Hiroshi Yoshida Koichi Nihei Kozo Satoda Dai Kanetomo
Cloud System Res. Labs., NEC Corp.
1753 Shimonumabe, Nakahara-ku,Kawasaki, Kanagawa, Japan
Email: {n-kai@cj, h-yoshida@jh, k-nihei@aj, k-satoda@cb, d-kanetomo@ce}.jp.nec.com

***Abstract*—Live streaming services over HTTP used on smart devices via a mobile network have become very popular. However, playback interruption often occurs during a live streaming. This is because TCP throughput fluctuates due to radio strength variation and cross-traffic, and the playback buffer on the client side is small. To solve the playback interruption problem, adaptive rate control techniques are helpful. However, using a smart device with insufficient CPU power leads to control delays, which cause the rate to be controlled improperly. This paper proposes a novel rate control method that uses plural codecs and detection of throughput degradation, and shows that the proposed method reduces playback interruptions on smart devices. We evaluate our proposed method with a live streaming for 100 seconds as an experiment in conditions in which the TCP throughput is fluctuated by a network emulator. Playback interruption occurred three times and lasted 1.8 seconds in total with the conventional method, but did not occur with the proposed method. We also evaluate the method from the viewpoint of QoE defined in ITU-T Rec.P.1201/Amd.2. The proposed method is able to improve QoE(MOS) to** 3.5 **from** 2.6 **of conventional method.**

I. Introduction

Live streaming services over HTTP used on smart devices via mobile networks have become very popular [1]. In a disaster or emergency, live streaming to a medical center helps to improve emergency operations [2], [3].

The major platforms for live video streaming are MPEG-DASH (Dynamic Adaptive Streaming over HTTP) [4] and Apple HLS (HTTP Live Streaming) [5]. These platforms use TCP/IP for video streaming. Since TCP throughput is fluctuated widely through a best effort network such as a mobile network, playback interruption often occurs when we watch live video streaming through a mobile network. Specifically, live video streaming needs a playback buffer on the client side to be made smaller because live streaming must emphasize a low latency. This may, however, cause frequent playback interruption.

To address these problems, adaptive rate control techniques have been developed [6]–[10]. The adaptive rate control techniques send video content whose bitrate (encode rate) dynamically adjusts to network throughputs. However, if the sender of an adaptive rate control system possesses insufficient CPU power (e.g., smart devices), it cannot switch the bitrate immediately in accordance with the switch request from the receiver. During video streaming, especially with a small playback buffer on the client side like in live video streaming, the delay from the time of bitrate switch request to the time of actual bitrate switching causes playback interruption.

This paper proposes a novel rate control on smart devices that uses plural codecs and a method for detecting throughput degradation. We evaluate the proposed method with the live video streaming for 100 seconds in conditions in which TCP throughput is fluctuates by a network emulator, and show that our proposed method can achieve higher quality than the conventional method.

This paper is organized as follows. In Section II, we briefly explain conventional adaptive rate control techniques. Section III describes the problems of conventional adaptive rate control on smart devices. In Section IV, we explain our proposed method. In Section V, we show the experimental results for a network emulator and QoE defined by ITU-T Rec. P.1201/Amd.2 [11]. Finally, we conclude this paper in Section VI.

II. Related Work

In this section, we briefly explain conventional adaptive rate control techniques. The techniques that switches the bitrate of content along TCP fluctuated throughput is called adaptive video streaming. The major examples of adaptive video streaming are Apple HLS (HTTP Live Streaming), Microsoft Smooth Streaming [12], and MPEG (Moving Picture Experts Group) MPEG-DASH (Dynamic Adaptive Streaming over HTTP). Shuai et al. [6] proposed that the server side should infer the playback buffer on the client side to decrease the delay of feedback between server side and client side. Lohmar et al. [7] showed the theoretical minimum delay of a HTTP live streaming was a five-segment-duration. De Cicco et al. [8] developed a method in which the server side calculates the throughput and reduces the playback buffer on the client side to 15 seconds. Bouzakaria et al. [9] developed the low delay live streaming with MPEG-DASH and GDR, but they didn't refer to the delay of a playback buffer on the client side. Akhshabi et al. [10] showed that the playback interruption occurred with Smooth Streaming when the length of playback buffer was 8 seconds.

Most adaptive rate control techniques switch the bitrate chunk by chunk, which is a part of a video file and lasts between several seconds and several tens of seconds. The authors consider that chunk-based rate control is also reasonable for live video streaming using a smart device. The reasons are

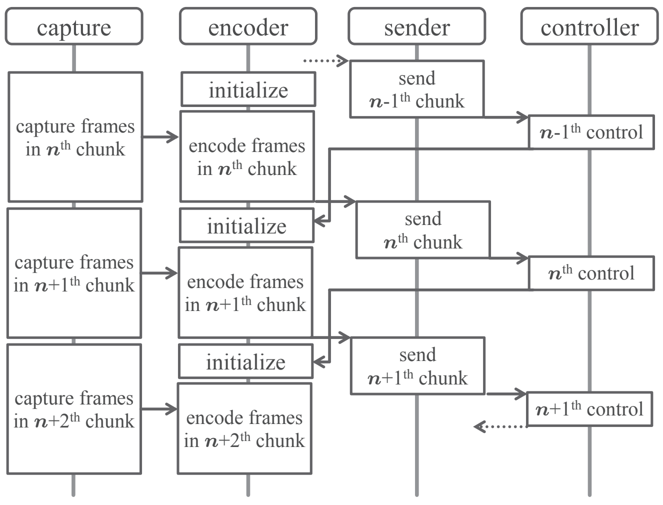

capture
encoder
sender
controller
initialize
send
n-1th chunk
capture frames in nth chunk
encode frames in nth chunk
n-1th control
initialize
send
nth chunk
capture frames in n+1th chunk
encode frames in n+1th chunk
nth control
initialize
send
n+1th chunk
capture frames in n+2th chunk
encode frames in n+2th chunk
n+1th control

Fig. 1: Delay in bitrate switching

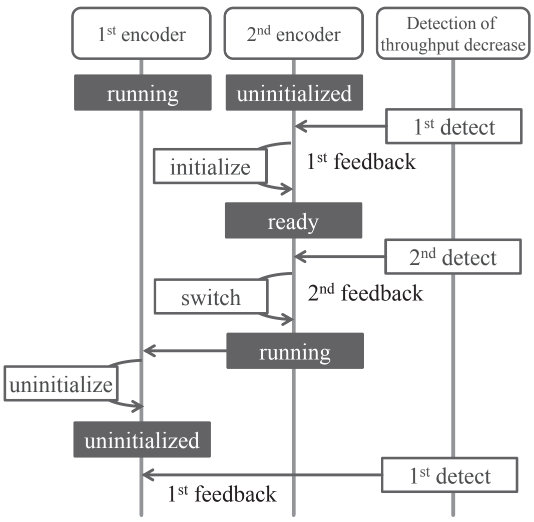


Fig. 2: Bitrate switching with multiple encoders

the bitrate switching needs a certain time space for resetting the encoder, (for example, several hundreds of milliseconds on a device with low CPU power such as a smart device), and frequent bitrate switching increased latency.

III. Problem of Adaptive Rate Control

In this section, we show the problems of conventional adaptive rate control on smart devices.

Figure 1 shows the flow of capturing, encoding, sending and bitrate switching for video streaming. Suppose the chunk contains multiple captured frames and bitrate is switched in each chunk. When the encoder starts to encode the $n^{\rm th}$ chunk, the controller orders the bitrate switch to the encoder and the encoder starts re-initialization. The actual encoding process starts after re-initialization. When the sender starts to send $n^{\rm th}$ chunk, the $n^{\rm th}$ bitrate control starts. However, at the point of the $n^{\rm th}$ bitrate control starting, the $n+1^{\rm th}$ chunk encoding has started already, so the $n^{\rm th}$ bitrate control affects the $n+2^{\rm th}$ chunk. As a result, $n^{\rm th}$ bitrate control is enabled after two chunks.

If we could switch the bitrate of the content during encoding, the bitrate control delay we stated above could be decreased. However, we cannot switch the bitrate of the content during encoding for two reasons.

First, the encoder initialization on the device that has a low power CPU needs a certain time. As we stated before, by using the ordinary codecs, we have to re-initialize the codec in order to switch the bitrate, and re-initializing the codec takes a certain time. The device we employed in the experiment is the 2013 model of Nexus 7, and the video encoder is H.264. Under these conditions, initialization takes $100 \sim 200$ milliseconds. Live video streaming requires low latency, which is why a chunk duration must be shorter such as 1 seconds. In this case, we have to initialize the encoder only once or twice each chunk. Otherwise, if we initialized the encoder three or four times in one chunk, we wouldn't achieve a real-time encoding for VGA resolution.

Second, the bitrate switching might increase the whole size of a chunk. In general, the head of the chunk is encoded as I-frame that can be decoded by itself, and the following frames are encoded as P-frames that refer to other frames. P-frame is usually smaller than I-frame. However, the first frame after the encoder re-initializes must be I-frame, so the frame that is encoded after switching the encoder could be bigger than the frame that was encoded before switching the encoder. We are concerned that the bitrate control doesn't work well because of the overhead we explained above. If the controller orders the bitrate switch in order to lower the bitrate, it may cause the playback interruption.

IV. Proposed Method

In this section, we explain our proposed method.

A. *Overview of Proposed Method*

We propose switching the bitrate by using the two encoders in parallel. Figure 2 shows the overview of our proposed method. On the left is the 1st encoder, in the middle is 2nd encoder, and on the right is the detection of throughput degradation, which is explained below. The 1st and 2nd encoders belong to the distributing device, and detection of throughput degradation belongs to the viewing device. While the 1st encoder is working, the 2nd encoder is still uninitialized. When the viewing device detects the throughput degradation, it sends 1st feedback to the distributing device. When the distributing device receives the 1st feedback, the 2nd encoder starts to be initialized. After that, when the viewing device sends 2nd feedback to the distributing device, the distributing device switches the encoder from the 1st encoder to the 2nd encoder and sets the 1st encoder as uninitialized.

Detection of throughput degradation is aimed at detecting the portent of throughput degradation. The reason the encoder switching needs two feedbacks is to achieve more accuracy. We explain detection of throughput degradation in details in IV-B.

When the distributing device switches the encoder, we have to consider the encoder switching overhead we explained before, so the decision of bitrate shift decides whether the encoder should be switching or not. We explain the decision of bitrate shit in details in IV-C.

Algorithm 1 Detection algorithm of throughput degradation

```
1:  for $t = \tau, 2\tau, 3\tau, \cdots$ do
2:    measure the cumulative data size received $R(t)$
3:    calculate the throughput $x(t) = (R(t) - R(t-\tau))/\tau$
4:    obtain the bitrate $b_c$ of the sending chunk $c$
5:    if $x(t) < b_c$ then
6:      if first time during sending chunk $c$ then
7:        conduct the 1st feedback
8:      else if second time during sending chunk $c$ then
9:        conduct the 2nd feedback
10:     end if
11:   end if
12: end for
```

With these systems, the distributing device can reduce the bitrate during the chunk. As a result, we can curtail the delay of bitrate control and decrease the frequency of playback interruptions.

B. Detecting Throughput Degradation

Algorithm 1 denotes the algorithm for the detection of throughput degradation. Suppose the sum of the size of the received video stream until the time t is $R(t)$ bit. The average TCP throughput each time interval τ is calculated to $x(t) = (R(t) - R(t-\tau))/\tau$. The time interval τ must be shorter than a chunk duration, so we set $\tau = 0.1$ sec. If b_c (the nominal bitrate of sending chunk c) is larger than $x(t)$, the algorithm detects the throughput degradation. If the algorithm has not detected the throughput degradation since the viewing device started to receive the chunk, the viewing device sends 1^{st} feedback to the distributing device. If the algorithm has detected the throughput degradation for the chunk, the viewing device sends 2^{nd} feedback to the distributing device. When the distributing device receives the 1^{st} feedback or 2^{nd} feedback, it proceeds as explained above.

C. Bitrate Shift Decision

Algorithm 2 denotes the algorithm of deciding bitrate shift. Suppose that the number of frames that should be contained in a chunk is N and that the distributing device receives the 2^{nd} feedback while the k^{th} frame is encoded. If the size of k^{th} frame that is encoded by the 1^{st} encoder is larger than the threshold X bit, it means the encoder switching overhead can be ignored, so the algorithm decides to switch at the k^{th} frame. However, if the number of skipped frames is larger than the threshold M or k equal to N, the algorithm decides to switch at the k^{th} frame. In this paper, we set $X = S_{\mathrm{I}}/2$ bit, $M = 5$. S_{I} is I-frame (the head size of the chunk). Bitrate shift decisions enable switching from the 1^{st} encoder to the 2^{nd} encoder at the frame that is relatively larger.

V. Evaluation

We conducted experiments of live video streaming for 100 seconds through a network emulator. In this section, we state the experimental results.

Algorithm 2 Decision of bitrate shift

```
1:  the sending chunk $c$ contains $N$ frames
2:  assume that the 2nd feedback is received at frame $f = k$
    $(1 \le k \le N)$
3:  for $f = k, \cdots, N$ do
4:    if size of frame $f$ encoded by 1st encoder $> X$ then
5:      conduct bitrate shift
6:      break for-loop
7:    end if
8:    if $f - k > M$ or $f = N$ then
9:      conduct bitrate shift
10:     break for-loop
11:   end if
12: end for
```

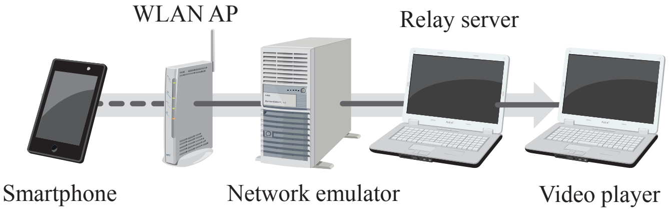


Fig. 3: Experimental environment

TABLE I: Details of experimental environment

Smartphone	Nexus7 2013(mobile model)
	Android 4.4
WLAN AP	Aterm WR8500N 802.11 a/b/g/n
Network emulator	Netem [13]
Traffic shaping	1.6[Mbps] (0∼30sec)
	800[kbps]∼1.6[Mbps] (30∼100sec)
Video player	Google Chrome 42.0 Flash Player

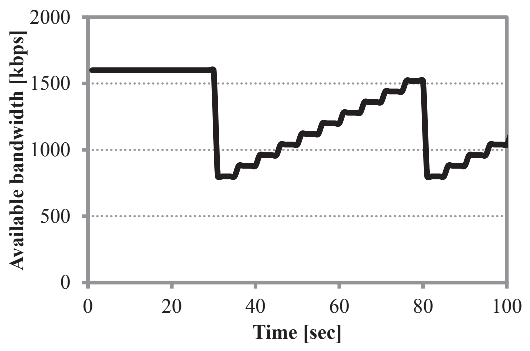
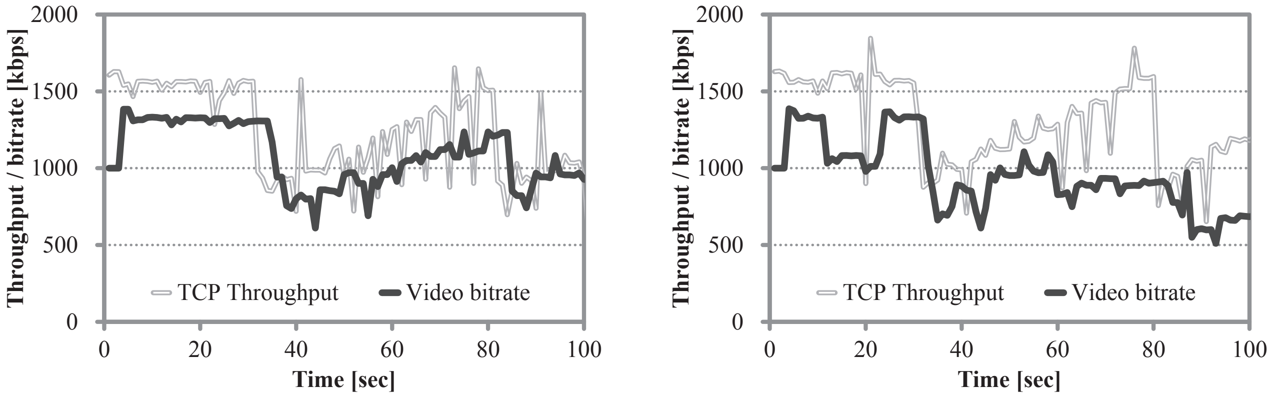


Fig. 4: Traffic shaping

A. Experimental environment

Figure 3 shows the overview of experimental environment. We conducted live video streaming from a chunk distributing device to a chunk viewing device for 100 seconds. The chunk distributing device and the chunk viewing device are connected by a LAN through a network emulator. The average elapsed time of codec initialization on the chunk distributing device is 100 milliseconds, and the range of a bitrate that is set to an encoder (encode bitrate) is 10 kbps $\sim$ 2.0 Mbps. An initial playback buffer time on the chunk viewing device is 1.0 seconds. If a playback interruption occurred due to



(a) Conventional

(b) Proposed

Fig. 5: Experimental results

the playback buffer on the chunk viewing device becoming empty, live video streaming delay was longer because the chunk viewing device played at the point of the playback interruption.

Figure 4 shows the traffic shaping graph by using Netem [13]. For the first 30 seconds of live video streaming, the available bandwidth is fixed at 1.6 Mbps. After that, the available bandwidth is raised linearly from 800 kbps to 1.2 Mbps. Once the available bandwidth equals 1.2 Mbps, the traffic shaping loosens to 800 kbps and is raised to 1.2 Mbps again.

We performed live video streaming of the conventional and proposed methods in the above environment. The conventional live video streaming just changes encode bitrates in accordance with feedback of TCP throughput from the chunk viewing device.

B. Results

Table II presents the average throughputs, video bitrates, playback interruption time, frequencies of playback interruption, and QoE of the conventional and proposed live video streaming methods.

The conventional video streaming achieved $1,079$ kbps of video bitrates on average under an average of $1,225$ kbps TCP throughputs. The proposed video streaming achieved 949 kbps of video bitrates on average under an average of $1,287$ kbps TCP throughputs.

In conventional live video streaming a playback interruption occurred three times for 1.8 seconds in total around 31 seconds after starting live streaming. On the other hand, no playback interruption occurred using the proposed method.

We also calculated QoE defined by ITU-T P.1201 using average video bitrate, playback interruption time, and interruption frequency of these results. The QoEs of the proposed and conventional methods are 3.61 and 2.56, respectively.

Figure 5a shows time sequence graphs of TCP throughputs and video bitrates using the conventional and proposed methods. Video playbacks using the conventional method stopped at 31 seconds due to quick decrease of the TCP throughput 30 seconds after starting.

TABLE II: Result of rate control

	conventional	proposed
throughput [kbps]	1,225	1,287
bitrate [kbps]	1,079	949
playback interruption time [sec]	1.8	0
# of interruption	3	0
QoE	2.56	3.61

C. Discussion

According to Fig. 5(a), the conventional method lowers video bitrate a few seconds after the sudden decrease of TCP throughput at 30 seconds. The conventional method takes time to follow TCP throughput as mentioned in Sec. III. This bitrate control delay causes the playback interruption one second after TCP throughput suddenly decreases. On the other hand, since the proposed method is able to follow TCP throughput immediately as depicted in Fig. 5(b), it avoids playback interruption due to sudden TCP throughput decrease.

Although another sudden TCP throughput decrease and bitrate control delay happened at 82 seconds, no playback interruption occurred a few seconds after 82 seconds. This is because the playback interruption that occurred at 31 seconds increased the playback buffer, i.e., live streaming delay. Although the playback buffer was 1.0 seconds long at the start of live video streaming, the playback buffer was 2.8 seconds long at 82 seconds because of playback interruption for 1.8 seconds at 31 seconds.

The average video bitrate of the proposed method is 12% lower than that of the conventional method from Table II even when the average throughputs of both methods are almost the same. This seems to be because the proposed method is sensitive to TCP throughput decrease and makes video bitrate decrease unnecessarily.

In spite of lower average video bitrate, the proposed method achieved higher QoE than the conventional one. This is because playback interruption deteriorates QoE much more than video bitrate. Therefore, the proposed method is able to increase live video streaming quality.

There are some points that the video bitrate does not lower despite TCP throughput degradation such as at around

20 seconds in Fig. 5(b). This is because the measurement time of TCP throughput and the video bitrate shift time are not synchronized precisely. Although the video bitrate shifts almost every 1 seconds, we get the TCP throughput data when every chunk data is sent completely. Hence the interval of TCP throughput value varies depending on the chunk size and TCP throughput. Since we determine the new bitrate in accordance with the latest TCP throughput data, video bitrate does not occasionally follow the TCP throughput fluctuation.

VI. Conclusion

Live video streaming using a smart device suffers from the degradation of the user-perceived quality due to frequent interruptions. The main reasons for the problem are insufficient CPU resources of a smart device (distributing device) and a small buffer on the viewing device. The former causes the control delay because of the initialization of the encoder and then triggers an underflow of the playback buffer in combination with the latter reason.

In this paper, we proposed a novel rate switching based on multiple encoders and the detection of the TCP throughput degradation. Our rate switching algorithm initializes or shifts the encoders on the basis of the feedback from the TCP throughput decrease detection, which reduces the rate control delay. The evaluation result shows that our method can avoid video interruption, whereas playback interruption occurs a total of three times (1.8 seconds) in the conventional method.

In our future work, we will experiment longer period via commercial mobile networks.

References

[1] [Online]. Available: http://www.ustream.tv

[2] A. Qiantori, A. B. Sutiono, H. Hariyanto, H. Suwa, and T. Ohta, “An emergency medical communications system by low altitude platform at the early stages of a natural disaster in Indonesia,” *Journal of medical systems*, vol. 36, no. 1, pp. 41–52, 2012.

[3] R. Handschu, R. Littmann, U. Reulbach, C. Gaul, J. G. Heckmann, B. Neundörfer, and M. Scibor, “Telemedicine in Emergency Evaluation of Acute Stroke Interrater Agreement in Remote Video Examination With a Novel Multimedia System,” *Stroke*, vol. 34, no. 12, pp. 2842–2846, 2003.

[4] I. Sodagar, “The MPEG-DASH Standard for Multimedia Streaming Over the Internet.” *IEEE Multimedia*, no. 18, pp. 62–67, 2011.

[5] R. Pantos, “HTTP live streaming,” 2014.

[6] Y. Shuai, G. Petrovic, and T. Herfet, “Server-driven rate control for adaptive video streaming using virtual client buffers,” in *Consumer Electronics– Berlin (ICCE-Berlin), 2014 IEEE Fourth International Conference on*. IEEE, 2014, pp. 45–49.

[7] T. Lohmar, T. Einarsson, P. Frojdh, F. Gabin, and M. Kampmann, “Dynamic adaptive HTTP streaming of live content,” in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2011 IEEE International Symposium on a*. IEEE, 2011, pp. 1–8.

[8] L. De Cicco, S. Mascolo, and V. Palmisano, “Feedback control for adaptive live video streaming,” in *Proceedings of the second annual ACM conference on Multimedia systems*. ACM, 2011, pp. 145–156.

[9] N. Bouzakaria, C. Concolato, and J. Le Feuvre, “Overhead and performance of low latency live streaming using MPEG-DASH,” in *Information, Intelligence, Systems and Applications, IISA 2014, The 5th International Conference on*. IEEE, 2014, pp. 92–97.

[10] S. Akhshabi, A. C. Begen, and C. Dovrolis, “An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP,” in *Proceedings of the second annual ACM conference on Multimedia systems*. ACM, 2011, pp. 157–168.

[11] *ITU-T Recommendation P.1201 Amd.2*, Std., 2012.

[12] A. Zambelli, “IIS smooth streaming technical overview,” *Microsoft Corporation*, vol. 3, 2009.

[13] S. Hemminger *et al.*, “Network emulation with NetEm,” in *Linux conf au*. Citeseer, 2005, pp. 18–23.