Proposal for ICME 2017 Grand Challenge:

Segment-based Rate Control of Video Encoder for Live ABR Streaming

Twitch Interactive Inc.

Contact: Dr. Yueshi Shen, Principal Research Engineer, yshen@twitch.tv

Background:

Live broadcasting of digital video has been in production since 1990s. The traditional linear TV (cable, satellite, terrestrial, etc.) typically simulcasts a transport stream to set top boxes that have limited memory. To avoid the playback of a video stream from encountering interruption, a video encoder needs to strictly follow the MPEG-2 VBV or H.264 HRD model. The idea is to limit the fluctuation of frame sizes within a specified buffer size, so the player never runs into buffer empty.

On the other hand, OTT live streaming services (e.g., Twitch, YouTube Live, Douyu, Huya, etc.) become more and more popular for the media consumer aged between 18 and 35. The rate control of Adaptive Bitrate Streaming (ABR) transcoder plays a critical role in determining the performance of ABR playback algorithms on any client platform (web, set top box, or mobile device). However, the OTT industry mostly uses video encoders with rate control algorithms developed many years ago for the linear broadcast, even though video is delivered in individual multi-second segments instead of a continuous bitstream. The traditional rate control methods work in general for ABR, however they can cause poor performance of ABR playback control, bringing bad QoS to viewers.

Problem Description:

In Adaptive Bitrate (ABR) streaming, a video stream is encoded in multiple bitrates (aka representations) for clients with different download bandwidths. In order for a client to smoothly switch among different representations in case its network condition changes, these bitstreams are segmented into multi-second parts. These segments have fixed durations and are perfectly time aligned vertically across multiple representations. To help ABR playback algorithms achieve their best performances, the result segment size of each representation ideally is equal to the segment duration times the advertised bitrate.

For example, a live source bitstream (e.g., 1080p60 at 4mbs) can be transcoded into 4 representations:

- 720p60, 2.0mbs
- 480p60, 1.0mbs
- 360p60, 0.5mbs
- 240p60, 0.25mbs

, and all of them in ***H.264 Main Profile***.

To reduce the end-to-end broadcast latency, every segment duration of both the source and the transcoded representations is fixed at ***2 seconds*** precisely. Therefore, the nominal segment sizes of the 4 transcoded representations are:

- ***720p60, 500k bytes***
- ***480p60, 250k bytes***
- ***360p60, 125k bytes***
- ***240p60, 62.5k bytes***

ABR streaming does not care about fluctuation of frame size within a segment, so an encoder doesn't have to follow the H.264 HRD model, as long as the final segment sizes are kept as close to the nominal values as possible. It is important to note that the effectiveness of an ABR encoder's rate control algorithm is judged by the final segment sizes, not the bitstream's compliance to H.264's CPB buffer model. On the other hand, dramatic fluctuation of quantization steps across frames should be suppressed to achieve good visual quality.

Since the application of this research work is for live broadcast, the ABR encoding has to take place in real time, therefore single pass encoding is prefered. Two passes is still acceptable, but the tradeoff between the rate control benefit and the extra computational cost needs to justified.

Also, transcoding latency should be kept as low as possible, so encoder using look ahead is allowed but the look ahead window should be kept within 1 second (30 frames).

Expected Research Outcome:

Twitch invites video researchers to develop new rate control algorithms for segment-based ABR streaming. A submission should include

- a detailed description of the algorithm, source code
- result and analysis of
  - segment sizes,
  - frame average/min/max Qp,
  - PSNR,
  - computational cost, and

a comparison against the reference result described below based on x264, or any result from your reference code base with its default setting (e.g., JM)

Award:

Based on the submitted research work's innovativeness, implementability and experiment result, Twitch will select 3 winners and offer certificates and money awards of

- 1st prize: $1000
- 2nd prize: $750
- 3rd prize: $500

Test Data:

Raw YUV 1920x1080 4:2:0 60fps, can be downloaded from https://media.xiph.org/video/derf/:

https://media.xiph.org/video/derf/twitch/y4m/CSGO.y4m
https://media.xiph.org/video/derf/twitch/y4m/EuroTruckSimulator2.y4m
https://media.xiph.org/video/derf/twitch/y4m/GTAV.y4m
https://media.xiph.org/video/derf/twitch/y4m/WITCHER3.y4m

Convert the y4m files into .yuv files:

ffmpeg -i CSGO.y4m CSGO.yuv
ffmpeg -i EuroTruckSimulator2.y4m EuroTruckSimulator2.yuv
ffmpeg -i GTAV.y4m GTAV.yuv
ffmpeg -i WITCHER3.y4m WITCHER3.yuv

Reference Result:

The reference result is generated by using ffmpeg (with x264 preset of medium).

First to build a customized libx264 (v20161024.2245) for ffmpeg (v3.1.5) on Linux:

1) *Download x264 and FFmpeg*
2) *After your modification within the x264 code base (e.g., in the encoder/ratecontrol.h or .c), go to the root directory of x264*
3) *Run "./configure --enable-static --disable-opencl --disable-lavf --enable-shared" without quotation marks*
4) *Run "make && sudo make install" without quotation marks*

5) *Then, proceed to go to FFmpeg root directory and run "./configure --enable-librtmp --enable-libx264 --enable-libmp3lame --disable-libspeex --enable-nonfree --enable-gpl --disable-shared" without quotation marks*
6) *Run "make && sudo make install" without quotation marks*
7) *Run "sudo ldconfig" without quotation marks*
8) *Now the FFmpeg will use the customized version of libx264*

Rate control decision and result can be printed from x264 rate controller's source code,

1) *Go to x264/encoder/ratecontrol.c*
2) *Go to method x264_ratecontrol_end(), this method is being executed after encoding one frame and it will save the statistics and update the rate control state*
3) *You can either change h->param.rc.b_stat_write to true to enable the stats printing or print the variable or results you need at the end of the method.*

Then executing the encoding by running the following FFmpeg commands, with scene change detection turned off and segment duration set to 2 seconds:

720p:
ffmpeg -f rawvideo -vcodec rawvideo -s 1920x1080 -r 60 -i "concat:CSGO.yuv|EuroTruckSimulator2.yuv|GTAV.yuv|WITCHER3.yuv" -c:v libx264 -b:v 2M -bufsize 2M -s 1280x720 -x264opts keyint=120:min-keyint=120:scenecut=-1 -profile:v main -preset medium result_720p60.ts

480p:
ffmpeg -f rawvideo -vcodec rawvideo -s 1920x1080 -r 60 -i "concat:CSGO.yuv|EuroTruckSimulator2.yuv|GTAV.yuv|WITCHER3.yuv" -c:v libx264 -b:v 1M -bufsize 1M -s 640x480 -x264opts keyint=120:min-keyint=120:scenecut=-1 -profile:v main -preset medium result_480p60.ts

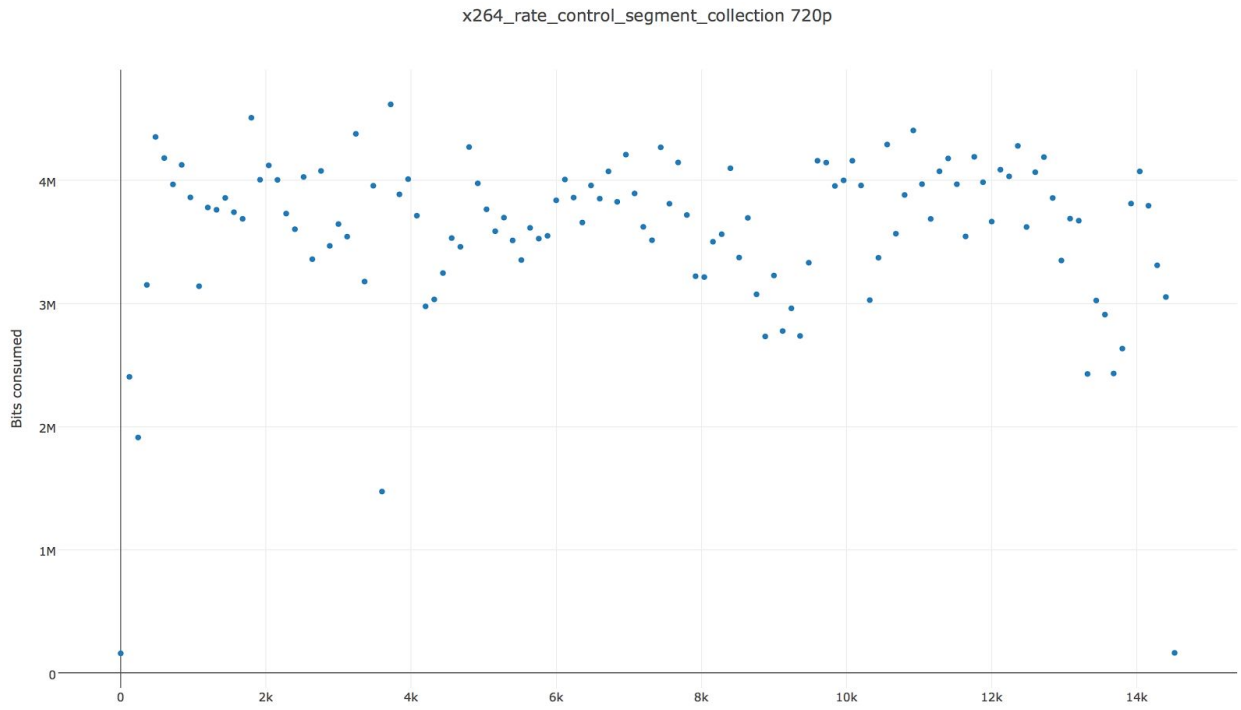
360p:
ffmpeg -f rawvideo -vcodec rawvideo -s 1920x1080 -r 60 -i "concat:CSGO.yuv|EuroTruckSimulator2.yuv|GTAV.yuv|WITCHER3.yuv" -c:v libx264 -b:v 500K -bufsize 500K -s 480x360 -x264opts keyint=120:min-keyint=120:scenecut=-1 -profile:v main -preset medium result_360p60.ts

240p:
ffmpeg -f rawvideo -vcodec rawvideo -s 1920x1080 -r 60 -i "concat:CSGO.yuv|EuroTruckSimulator2.yuv|GTAV.yuv|WITCHER3.yuv" -c:v libx264 -b:v 250K -bufsize 250K -s 352x240 -x264opts keyint=120:min-keyint=120:scenecut=-1 -profile:v main -preset medium result_240p60.ts

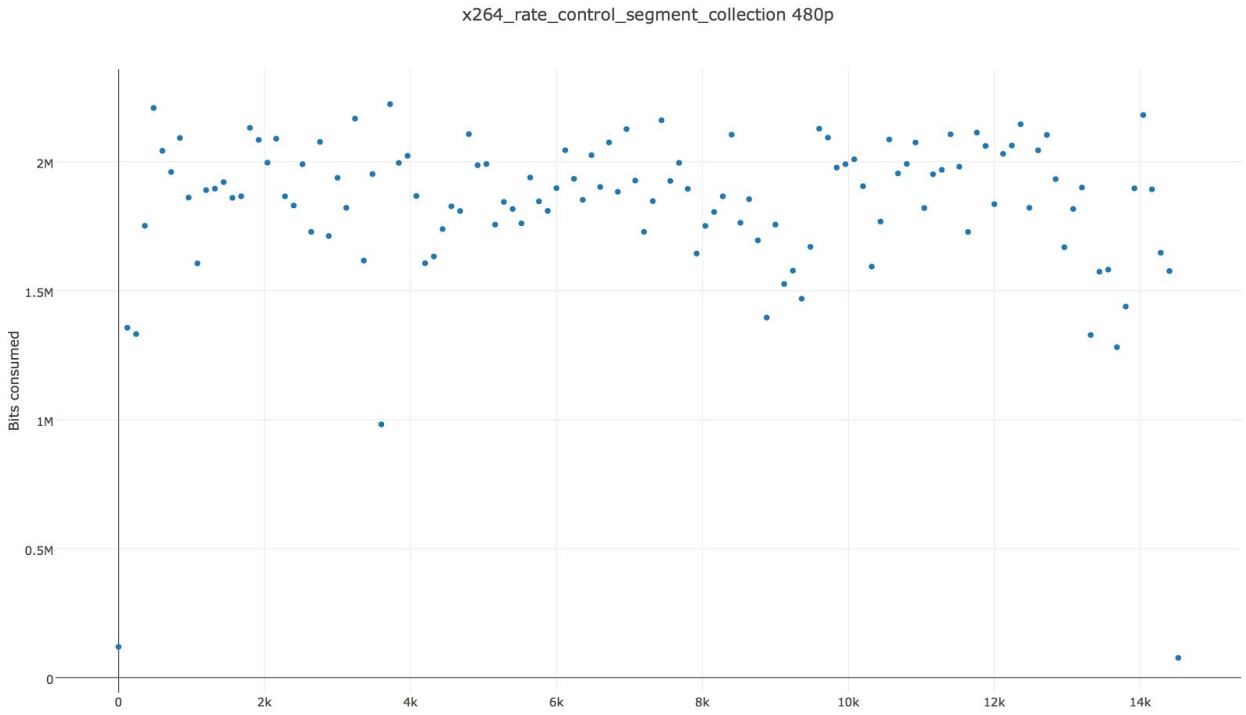
The reference results from the above x264 configuration and the segment sizes fluctuate quite a lot. Result TS files and segment size plots can be downloaded at

https://www.dropbox.com/sh/wy9a5nnnyat5q56/AADTsRHRiVol0XRajQCX3H5wa?dl=0

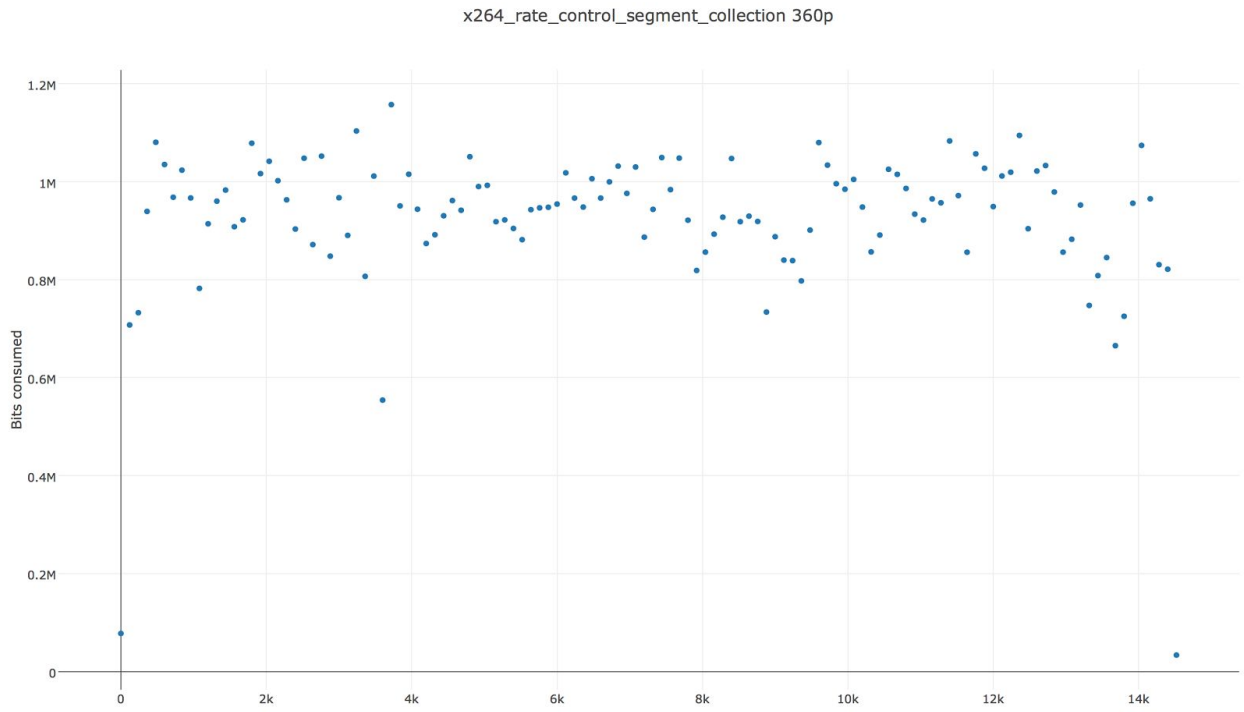
720p60 (nominal segment size 500K bytes or 4M bits):



480p60 (nominal segment size 250K bytes or 2M bits):



360p60 (nominal segment size 125K bytes or 1M bits):



240p60 (nominal segment size 62.5K bytes or 500K bits):