Improving Quality of Experience for Mobile Broadcasters in Personalized Live Video Streaming

Qingmei Ren[1], Yong Cui* [1], Wenfei Wu[1], Changfeng Chen[1], Yuchi Chen[2], Jiangchuan Liu[2], and Hongyi Huang[1]

[1]Department of Computer Science and Institute for Interdisciplinary Information Sciences, Tsinghua University
[2]Department of Computer Science, Simon Fraser University

Abstract

Ensuring high video quality of experience (QoE) on the broadcaster side is critical for interactive live streaming. However, measurements on multiple live streaming platforms show that they all suffer from broadcaster-side video quality degradation in the presence of transient bandwidth fluctuations. This paper presents *Greedy Variable Bitrate* (GVBR), a suite of solutions that optimizes the QoE through an approriate keyframe interval that trades cross-frame compression for lowered inter-frame interdependency, a simple-yet-efficient frame dropping strategy to prevent excessive frame drops, and a bitrate adaptation strategy customized for broadcasters who have shallow buffer. We compare GVBR with state-of-art algorithms in different network conditions, and find that GVBR can cut video interruption incidents by 90%, while achieving comparable bitrate.

I. Introduction

Recent years with more personal devices equipped with high-definition cameras, applications that allow device users to stream videos to anyone get a rapid proliferation (e.g., Facebook Live, Periscope, Twitch, Douyu). While recent work on personalized live streaming has insofar focused on analyzing its traffic pattern (e.g., [21]) and video distribution architecture (e.g., [14], [17]), there has not been enough effort to characterize the quality issues of *broadcaster-uploaded videos*. Yet, we argue that *improving the broadcaster-side uploading video quality is crucial to the QoE in the personalized live streaming.* Any delay or failure caused by the broadcaster could inflate on all viewers. Moreover, the upstream video quality sets a "cap" on all viewers.

Compared with traditional live streaming where broadcasters have well-provisioned connections and streaming delay is tens of seconds, the broadcaster-side video uploading in personalized live streaming distinguishes itself in two aspects. *(1) Individual mobile users as broadcasters.* Content source in personalized video streaming is often mobile broadcasters that encounter a more variable network connection (e.g., user motion and complicated wireless signal strength).

Supported by the Science and Technology Project of State Grid Corporation of China (No.2017YFB1010002) and Funding of Beijing National Research Center for Information Science and Technology.

*Corresponding author: Yong Cui (cuiyong@tsinghua.edu.cn)

(2) Broadcaster-viewer interactivity. The broadcasters are required to interact with viewers, and thus the end-to-end streaming delay must not exceed several seconds.

Due to its unique characteristics, there are three requirements on the broadcaster streaming protocol design.
1. High quality. Video quality includes many metrics, including bitrate, frame-per-second (FPS), resolutions. Broadcasters must provide videos with as highest possible quality as possible and then the viewers can get satisfactory QoE.
2. Agile adaptation. The broadcaster must be sufficiently adaptive to quickly react to bandwidth fluctuations.
3. Timeliness. The delay between the broadcaster and viewers must be minimized or at least bounded.

Directly applying frameworks to traditional live streaming leads to the QoE of broadcaster-uploading video far from ideal. We observe two *prevalent* quality issues across many popular platforms. (1) An *amplifying effect of transient fluctuating network conditions* causes persistent video QoE degradation. (2) These broadcasters are unable to effectively respond to long-term throughput drops as well. Thses problems cause significant quality degradation in practice, especially for the broadcasters that are often subject to both transient and long-term wireless throughput fluctuations.

The root cause of amplifying effect lies in the fact that RTMP drops frames too aggressively when the buffer overflows, resulting in unnecessary drops of video frames and persistent video stalls. RTMP, the de-facto streaming protocol, is widely used in many popular platforms; alternative HTTP-based live streaming protocols have also been studied [12] [19]. While switching to HTTP-based protocols might achieve better video quality, it cannot timely react to wireless fluctuation due to chunking overheads.

For the long-term network fluctuations, existing solutions focus on the bitrate adaptation in video-on-demand (VOD), which can be classified into three categories: rate-based, buffer-based and the combination of two. Rate-based methods pick the highest available bitrate lower than the estimated bandwidth [9] [16], while buffer-based chooses the bitrate according to the buffer [8]. Control theory is also applied to the bitrate adaptation, which uses the combination of buffer and throughput [20]. The difference between VOD and live streaming are the chunk granularity and the buffer size. There

978–1–5386–2542–2/18/$31.00

are also some papers about video adaptation in live streaming, but little talks about video transmission quality. Cicco et al. [6] use feedback control to switch the encoding parameter, but the issue lies in the server-client distributing link.

In this paper, we present GVBR, a suite of solutions that substantially improves the broadcaster-side video quality. Our key insight is that these quality issues can be mitigated by a systematic co-design of RTMP configuration (i.e., keyframe interval, buffer size), frame-level drop logic, and higher-level bitrate adaptation strategy. While integrating GVBR in existing broadcaster involves changes in multiple aspects, all changes are non-intrusive, changing either tunable parameters or control logic that is not hard-coded in the software.

Our preliminary evaluation shows that a better design could obviously improve video quality compared to an open-source RTMP platform. Through extensive evaluation under a variety of network conditions, we find that GVBR cuts video interruption incidents by 90% compared to popular video adaptation algorithms, while achieving comparable bitrate.

In short, we make two contributions:

1) We are the first to shed light on the broadcaster-side video quality issues across personalized live streaming. Measurement results reflect a prevalent quality issue, caused by unnecessarily persistent video interruptions in the presence of transient bandwidth fluctuations.
2) We present a holistic suite of solutions that systematically address the observed quality issue via better designs for the encoding of frames, frame prioritization strategies, as well as bitrate adaptation strategy.

II. Video Quality Issues on Broadcaster Side

We measure several personalized live streaming platforms and show the broadcaster-side quality degradation issue. We then analyze the root causes, and propose possible solutions.

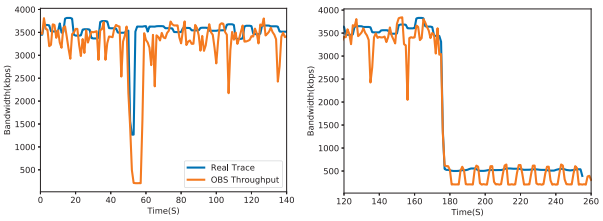
A. Measuring Broadcaster Performance

Video streaming architecture. A brief introduction to the common architecture of personalized live streaming is as follows. When live streaming starts, the broadcaster uploads the live video to a source server using RTMP protocols, from which the video is forwarded to many CDN edge servers. Then, each viewer streams the video from a nearby edge server using HTTP-based streaming protocols (i.e., DASH).

Experiment setup. We set up a demo that a broadcaster uploads video. Broadcaster and viewer are equipped with 100Mbps NICs; they are connected by a switch. The broadcaster uses OBS studio [3] (one of the most popular broadcast softwares) to send videos, and it has a bandwidth control module to emulate network bandwidth in wireless environments. The viewer is built on nginx-rtmp module to receive videos and uses VLC player to play the rtmp stream.

Case study: performance in variable networks. We control the network bandwidth according to a real trace from a wireless network [4]. The result is shown in Figure 1.

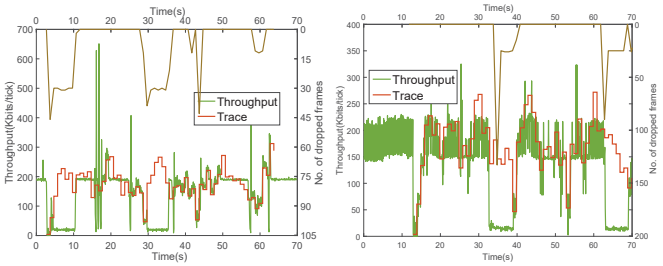
We have two observations. (1) In Figure 1a, the actual throughput follows the trace closely. However at 50s, the



(a) Throughput of $0-140$s (b) Throughput of $120-260$s

Fig. 1: Streaming throughput in oscillating wireless network. The trace records the real-time bandwidth when a mobile device user visits Amazon website. We aggregate packets into 5s bins and calculate the data amount in each slot. We stream video at a bitrate of 3300kbps (below the average bandwidth) via OBS and capture packet trace in viewers' side.

bandwidth falls below the bitrate for 2s, while the throughput degrades to almost 0 from 50s to 58s. It's an abnormal behavior, as *a 2s network jitter cascadingly causes 8s throughput falling in application level.* (2) Live streaming platforms cannot efficiently handle the long-term bandwidth variance in Figure 1b. Bandwidth drops dramatically after 180s and the period lasts for $80s$. In the challenging network, OBS insists the previous bitrate and massive frame drops occur.



(a) OBS player to Douyu server (b) Douyu tool to Douyu server

Fig. 2: Throughput and No. of dropped frames in commercial platforms. We repeat the above experiement with two settings: (1) use OBS to stream to Douyu servers, (2) from Douyu tool to Douyu server. In both settings, the average available bandwdith (1700kbps) is above the video bitrate, while intermittently, the throughput drops below the bitrate. Frame drops occur and keep a high value in certain periods, e.g., 30-36s in OBS player, 32-39s in Douyu tools.

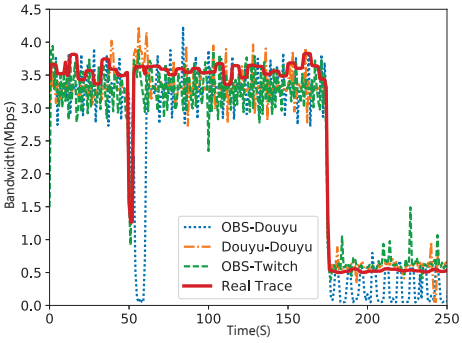


Fig. 3: Throughput in long-time bandwidth drop case. Bandwidth drops dramatically after 180s. During this period all the video bitrate keeps identical.

Experiments on commercial platforms. The "cascading effect" of the quality issues is prevalent, which manifests itself in many commercial platforms of personalized live streaming, as shown in Figure 2. We also find that the

cascading effect is not related to the instantaneously available bandwidth. For example, in Figure 2a, a dramatic bandwidth drop at 30s causes the cascading effect; while in Figure 2b, only a slight bandwidth drops at 32s.

The ability to handle long-term throughput drop is validated in Figure 3. Play failure time equals to the time corresponding to frame drops. OBS to Douyu performs worst, with the play failure time equaling to 93.1s. Others though follows the bandwidth, frames are dropped constantly in this period. Obs player to Twitch has a 79.3s play failure time, while Douyu tools to douyu has a 66.67s play failure time. In all, present commercial platforms cannot solve the long-term throughput drop, due to the excessive data generation speed.

B. Analyzing the Root Cause

The "cascading effect" arises from the inter-frame dependency. In case of bandwidth drop, there exists a queue to store video frames. A frame generating thread encodes raw captured images into frames, and enqueues the frames. According to H.264, the de-facto video encoding standard, a video is organized into several groups of pictures (GOP). In each GOP, the first I frame keeps unchanged; P frames are generated by computing their delta with the preceding I or P frame; B frames are computed based on its neighboring frames. Meanwhile a sending thread sends frames out via socket. If in bad network conditions, the frame sending thread would be blocked, disabling the generating thread to enqueue frames and thus dropping them. And if frame drops occur in the beginning or middle of a GOP, it would cascadingly render the remaining frames with the GOP undecodable.

We study the frame drop strategy in OBS. $Drop\ priority$ is designed to avoid the useless transmission of undecodeable frames. At first, drop priority is set as false, and would become true when frames drop. Then the queue stop receiving the subsequent frames until next GOP. $Timespan$ represents the maximum time difference among frames in the buffer. When a new frame arrives, if it's an I frame, it is enqueued. If the frame is a P frame, only if the timespan is less than 0.9s and the drop priority regarding P frame is true, the P frame is enqueued; otherwise, all frames within the GOP are dropped (except I frame). If the frame is a B frame, the threshold is 0.7s, and the processing logic resemble that of P frames.

However, in long-term bandwidth degradation, frame generation speed exceeds the network capacity, and frames are dropped. The cascading effect would further worsen the performance. The essential reason is that the video bitrate cannot be adjusted adaptively to the instantaneous bandwidth.

C. Design Space Insight

Massive frame drops obviously violate the quality requirements, but increasing video buffer would exceed the timeliness limitation. Thus in our design, we keep the buffer as $0.9s$ to satisfy the timeliness requirement, and adapt the frame generation mechanisms to variable network conditions. The previous motivating case gives us three possible methods.

Reduce the dependency between frames. The dependency between frames is due to the video compression algorithm, where compressed frames are computed from keyframes. Keyframes are independent. Thus, the dependency is limited within each GOP, the intuition to eliminate dependency is to reduce GOP size. Nevertheless, this approach may be a tradeoff between the minimal frame drop and the video quality, because reducing keyframe interval means less compression in video streaming. To reach a preconfigured bitrate, per-image quality would be degraded.

Drop frames wisely. Frame-dropping policy could strike a balance between quality and timeliness in the presence of bandwidth fluctuation [7], [10], [15]. Default OBS drops all P/B frames when the buffer exceeds a threshold. It is kind of reasonable because if dropping the earlier frames, the following frames cannot be decodable. And if dropping the latest several frames, the timeliness will be violated. But intuitively, for the case where two GOPs coexist in the buffer, dropping the frames of the old GOP, may have better performance. It is challenging to design an online frame dropping strategy that approaches the optimal solution.

Adaptive bitrate. Bandwidth fluctuations occur frequently, and measurements show that commercial applications use only constant bitrate (CBR). These methods cannot follow the dynamic bandwidth, which would bring tremendous frame dropping, even worse if the bandwidth drop lasts for a while. One possible and efficient solution is applying adaptive bitrate to broadcaster's side.

III. GVBR Solution

Based on the intuitions above, we customize the design of RTMP protocol in three aspects. First, we tune the GOP size to reduce the dependency between frames so that "cascading effect" is mitigated; second, we improve the frame drop strategy within GOP to avoid unnecessary frame drops. These two methods are targeted at short throughput drop. We also devise a GOP-level bitrate adaptation algorithm for long-term wireless bandwidth degradation.

A. Determining GOP Size

Larger GOP size causes "cascading effect" when the network suffers from transient bandwidth drop; but smaller GOP causes high compression ratio and subsequently low frame resolution. Here, we try to find the best GOP size.

The influence of GOP size on frame drops is measured by controlled experiments. We replay the stored video with controlled network conditions, tune GOP size in each replay, and figure out the GOP size with the least frame drops. As for the video resolution, SSIM(structural similarity), an effective method for measuring quality of video [18] [13], is used as the metric. Then we vary GOP sizes, and figure out the minimum GOP size that can keep SSIM within [95%-100%] of the SSIM of the original video. This computation can be done offline in a cloud, CDN server, or in the streaming device.

B. Smart Drop Strategy

In the runtime with GOP size configure, a frame drop logic is needed for transient network degradation. We first

maximize $\Sigma_i y_{iT}$, subject to	
$x_{ij} + y_{ij} + z_{ij} = 0, \forall j < i$	(1)
$x_{ij} + y_{ij} + z_{ij} = 1, \forall j \geq i$	(2)
$x_{ij} \geq x_{i,j+1}, \forall j \geq i$	(3)
$y_{ij} \leq y_{i,j+1}, \forall j \geq i$	(4)
$z_{ij} \leq z_{i,j+1}, \forall j \geq i$	(5)
$y_{ij} = \max\{1, 1 - z_{i,j-1}\}, \forall j, i \leq M_j$	(6)
$y_{ij} + z_{ij} = 1, \forall j > i + T_2$	(7)
$y_{i+1,T} \geq y_{iT}, \forall i \not\equiv N - 1 (\text{mod} N)$	(8)

Fig. 4: Frame Drop Strategy

theoretically figure out the best possible drop decision (i.e., best video quality) assuming the network degradation is known beforehand. Then we design an online algorithm that has low complexity and is suitable for mobile devices.

1) Problem Analysis: Assuming the frame pattern of GOP and network bandwidth are known, there exists an optimal scheduling maximizing audience QoE within the system constraints. A GOP comprises three kinds of frames, namely I/P/B frame; for convience, B frame is ignored. The problem can be formulated by integer programming (Figure 4). We divide the total decision time into T slots, and assume the i-th frame is generated at time i. We define x_{ij}, y_{ij}, z_{ij} as 0/1 variables to describe in time j whether i-th packet is in the queue, sent or dropped. At time j, the bandwidth capacity is C_j. Besides, the keyframe interval is defined as N.

Frame conservation constraints. Frame i is generated at time i, and after that, it is either in the queue, sent, or dropped (1-2). After a packet is removed from the queue, it would never be enqueued (3). After a packet is sent/dropped, it is permanently sent/dropped (4-5).

Bandwidth constraints. Here we assume that the broadcaster sends as many packets as possible when meeting the network capacity constraints. In addition, the frame that can be send must be in the buffer (6). Considering the constraints, the max sendable frame index M_j, is calculated by the function.

$$M_j = argmax \Sigma_k (1 - y_{k,j-1})(1 - z_{k,j-1}) \leq C_j \qquad (1)$$

Timeliness constraint. A frame can be "fresh" within $T1$ since its generation, after T_1 it's either sent or dropped (7).

Decodability constraints. The final delivered frames must be decodable. I frames are always decodable. A P frame is decodable only if its preceding frame is decodable (8).

Optimization goal. The goal is to maximize the delivered frames. Compared with prior work [15], this IP model has timeliness and decodability in consideration.

2) Greedy Algorithm: IP can achieve the offline optimal. Nevertheless long-term bandwidth is unknowable ahead of time, and the computational complexity is too high for mobile devices. Consequently, an online drop strategy is necessary.

Considering the situation where two GOPs coexist in buffer, we propose a modified dropping algorithm, called GreedyDrop. Differing from dropping all the P frames in the buffer by default, GreedyDrop drops all the P frames until the next keyframe. Hence the latest GOP can be reserved and our algorithm avoid frame dropping at least one GOP.

C. Adaptive Bitrate

maximize $\sum R_j - \alpha \sum \lvert R_{j+1} - R_j \rvert - \beta \sum D_j$, subject to	
$R_{j+1} = R_j, \forall mod(j, M) \not\equiv M - 1$	(1)
$S_j = argmax \sum_k R_k^j * T_k^j \leq C_j, \forall j$	(2)
$Rest_j = (C_j - \sum_{S_j} R_k^j * T_k^j)/R_{S_j+1}^j, \forall j$	(3)
$F_j = sgn(\sum_{S_j+1} T_k^j - Rest_j - T_1), \forall j$	(4)
$D_j = F_j * (T_{S_j+1}^j - Rest_j), \forall j$	(5)
$N_{j+1} = N_j - S_j - F_j + 1 - sgn(mod(j, M)), \forall j$	(6)
$R_k^{j+1} = R_{k+S_j+F_j}^j, \forall j, k \in \{1, N_j - S_j - F_j\}$	(7)
$R_{N_j - S_j - F_j + 1}^{j+1} = R_{j+1}, \forall mod(j, M) \equiv 0$	(8)
$T_k^{j+1} = T_{k+S_j+F_j}^j, \forall j, k \in \{1, N_j - S_j - F_j\}$	(9)
$T_{N_j - S_j - F_j}^{j+1} = T_{N_j - S_j - F_j}^{j+1} - D_j - Rest_j, \forall j$	(10)
$T_{N_j - S_j - F_j + 1}^{j+1} = 1, \forall mod(j, M) \equiv 0$	(11)

Fig. 5: Video Adaptation Formulation

1) Problem Formulation: To deal with long-term bandwidth fading, we introduce the adaptive bitrate. Symbols C_j, j, T, $T1$ are defined the same as the previous section. R_j represents the bitrate of the j frame. F_j and D_j describe whether frame drops occurs and the number of dropped frames. $Rest_j$ is the remaining time in buffer. Adopting GreedyDrop as frame drop strategy, the utility function (QoE) can be formulated as in Figure 5. The first item R_j is the bitrate utility, the second represents the bitrate switch penalty, the last one equals the frame drops penalty. Variables α and β are the utility parameters of bitrate switch and frame drops.

Bitrate Constraint. Constraint (1) requires that bitrate within one GOP must be identical.

Bandwidth Constraint. The maximum number of GOPs that can sent within the limited bandwidth is S_j (2).

Timeliness Constraint. Constraint (3) judges whether the remaining time after sending exceeds the buffer threshold and constraint (4) gives the number of dropped frames in time j.

State Transition. At time j, the total number of GOPs is N_j; for the k-th GOP, the remaining time and bitrate are T_k^j and R_k^j respectively. Constraints (7-11) reflect the state transition of the bitrate and remaining time of the GOPs in the buffer. Equation (6) describes the number of GOPs in time slot $j+1$, and the last items $1 - mod(j, M)$ represents whether the $j-th$ frame is the keyframe. sgn is the modified sign function. When the variable is greater than 0, it equals to 1; otherwise it equals to 0. mod is the modulo operation.

Offline optimal solution is hard to calculate. Assume for each GOP, the broadcaster can choose one from total M bitrate candidates. For a t GOP decision, the computation complexity equals to M^t, which has an exponential complexity.

Algorithm 1 Greedy Variable Bitrate(GVBR) algorithm

```
Initialize Rest=0, Send=0, Drop=0, η
for j=1 to T do
    record the history bandwidth [C_{j-τ}, C_{j-1}], use harmonic mean to estimate C_j
    choose the closest bitrate R_j to (C_j − rest)/η
    send frames Send in buffer within the bandwidth limit
    judge whether to drop extra frames Drop
    calculate the rest data size in buffer Rest = Rest + R_j − Send − Drop
```

2) Solution Description: The offline optimal with exponential complexity is hard to solve and on the basis of off-the-shelf knowledge of future bandwidth. Long-term bandwidth

prediction is inaccurate, and an intuitive idea is to change the bitrate following the bandwidth. Moreover, the remaining data size in the buffer can also be adopted. GVBR is designed for broadcaster's bitrate adaptation, as shown in Algorithm 1.

1. Bandwidth estimation. According to Festive [9] and MPC [20], harmonic mean is a useful method of estimating the future bandwidth. Here we use harmonic mean.

2. Bitrate selection. Given the future bandwidth C_j and the data size in buffer $Rest$, assuming the tuning parameter as η, choose the largest available bitrate lower than $(C_j - Rest)/\eta$.

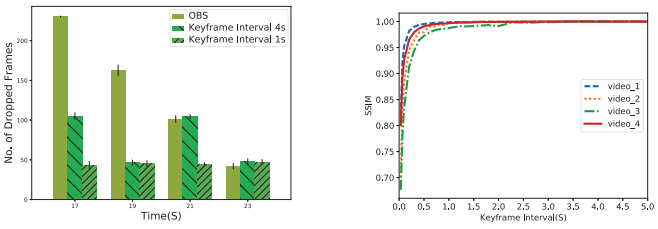
IV. Evaluation

Evaluation of the previous design in RTMP protocol is shown in this section, so does the GVBR algorithm.

A. Best GOP

In this section we evaluate reducing keyframe interval to figure out whether it can reduce the frame dropping.

Implementation. We keep the outbound throughput of broadcaster at a constant level, and introduce a 2s interruption. We record the number of frame drops as comparison metrics. The FPS in this paper is usually set as 30.



(a) Frame drops with different I frame interval. Default interval is 8s, and change it to 4s and 1s.

(b) SSIM for different GoP values. We pick four videos from the dataset to represent the result.

Fig. 6: Relationship between GOP size and video quality

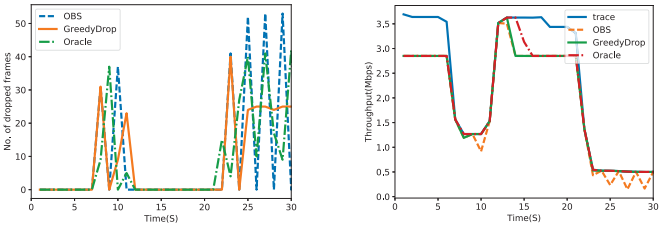
Varying keyframe interval. The frame drops are shown in Figure 6a. We can observe that in each individual experiment, earlier the interruption starts in a GOP, more frames are dropped. Because an early frame has more frames depending on it. For the default keyframe interval, when interruption starts at 17s, 19s, 21s, and 23s, the number of frame drops is 238, 164, 105, and 48. Comparing bars within the group of 17s, we find that smaller keyframe interval significantly reduces the number of frame drops (i.e., 238, to 102, and 46 when the interval changes from 8s to 4s and 1s).

This experiment shows that if we eliminate the dependency between frames, an occasional network jitter would only affect frames within a limited duration near the jitter. However, reducing keyframe interval is an intractable issue because that adjusting would cause video quality degradation.

Video quality and GOP To figure out the relationship, we vary the GOP sizes and repeat encoding uncompressed video using x264 encoder [5] [11]. A fact is that x264 uses the delta intermode coding. Thus a larger GOP is more likely to increase the cumulative error, and the recommended value of GOP is less than 250 frames. Nevertheless how to determine the specific value is still sophisticated. The video dataset we use contains SD, HD, and 4k content in variety [2]. The relationship between normalized SSIM and GOP size is displayed in Fig 6b. From the figure, we can see when the GOP size is larger than 0.5s, the SSIM keeps almost the same. Combining the previous experiment, we can see that GOP size between $[0.5, 2]$s may be the best choice.

B. Greedy Drop Strategy

We compare the performance with two drop strategies, Oracle and Default OBS. Oracle, the offline optimal solution by brute-force searching, has an exponential time complexity.



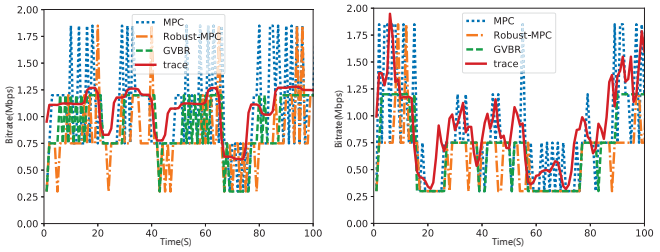
(a) No. of frame drop

(b) Real-time throughput

Fig. 7: Comparison of different frame drop strategies. We pick out 30 seconds from the dataset. During the period both long-term and short period bandwidth fluctuating appear. The keyframe interval is chosen between [0.5s,2s], 1s.

OBS drops the most frames (320 frames), and GreedyDrop reduces 15% (274 frames), which is a prominent improvement; and the gap between GreedyDrop and Oracle (265 frames) is small, which is less than 5%. The frame drop and throughput is shown in Figure 7. Frame dropping happens at 5-10s and 20-30s. All algorithms perform similarly during 5-10s, but the Oracle saves more frames before the network recovers, and keeps a high bandwidth during 10-15s. The frame drop of OBS waves at a high variance in 20-30s, but GreedyDrop almost keeps unchanged, because GreedyDrop only drops the undecodable frames of the first GOP. Oracle also fluctuates, but with a small variance. Considering time complexity and performance, GreedyDrop is a good choice.

C. Greedy Adaptive Bitrate



(a) FCC dataset [4]

(b) HSDPA dataset [1]

Fig. 8: Throughput of different datasets. GreedyDrop is adopted as the drop strategy (excluding OBS-VBR).

Three algorithms work excellently in VOD scenario is used for comparison, harmonic mean is used for bandwidth estimation:

- OBS-VBR: A simple but popular method, chooses the highest bitrate lower than the estimated bandwidth.

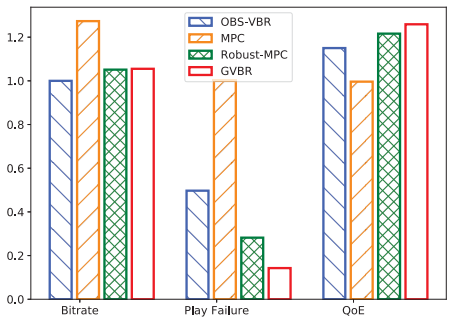


Fig. 9: Normalized bitrate, play failure and QoE

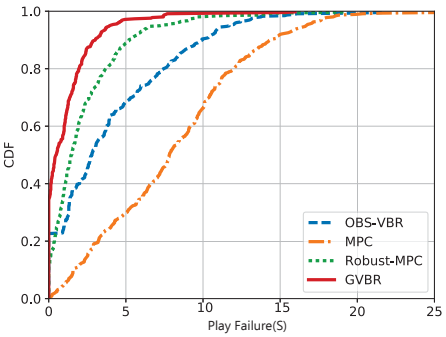


Fig. 10: CDF of Play Failure Seconds

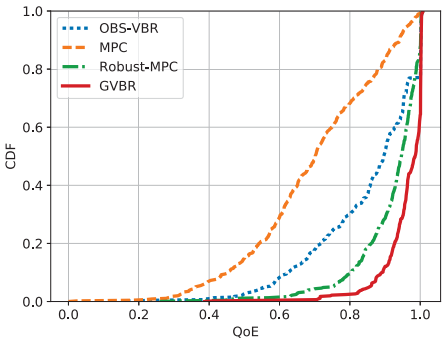


Fig. 11: CDF of Normalized QoE

- MPC (Model predictive control) [20]: use buffer state (buffer size, frame types) and bandwidth predictions to calculate the optimal bitrate operation in future several time slots. Use the first bitrate choice in next time slot.
- Robust-MPC [20]: use the approach resembling MPC, but correct the estimated bandwidth by considering the prediction error in the past several time slots.

Detailed results are shown in Figure 8. Without predication correction, MPC prefers to choose higher bitrate than Robust-MPC. Additionally, the MPC and Robust-MPC switch bitrate more frequently than GVBR. Because GVBR tends to choose the lower bitrate than the throughput, when small bandwidth fluctuation happens, GVBR is less likely to shake. However MPC struggles to achieve the optimal utility, and has the potential to choose a higher bitrate to maximize bitrate utility.

We use two datasets, FCC and HSDPA to evaluate GVBR algorithm. Massive simulation is displayed in Figure 9. Normalized QoE utility is compared in the figure. MPC has the maximum average bitrate, because the bandwidth estimation is more aggressive. Others reach almost the same average bitrate, with little difference, but GVBR is a little higher. With higher bitrate, MPC also drops the most frames, and the time of play failure is the longest. GVBR reduces the play failure to a small value, which is a 50% reduction compared with Robust-MPC. With higher bitrate and lower play failure, GVBR definitely preforms the best, with the highest QoE.

CDF about play failure and normalized QoE is in Figure 10, 11. In GVBR, 98% of the paly failure is less than 5s, 40% even plays fluently with no failure; and only 2% receives poor QoE, and the the rest 98% receives at least 80% normalized QoE. The total frame drops compared with original OBS with CBR are reduced by 96%. The play failure time of original OBS is 26s in average, and GVBR is only 1s.

V. Conclusion

We proposed GVBR, a combined algorithm which improves the default frame dropping strategy and design an effective video adaptation algorithm. The revised frame dropping strategy considers the case where two GOPs exist in the buffer and keep the frames of the next GOP, and thus have only 5% more frame dropping than Oracle. GVBR chooses the bitrate according to the difference between the estimated bandwidth and the data size in buffer, which reflects more accurately the real bandwidth. Massive experiments illustrates that GVBR reduces 50% of frame dropping compared with the state-of-art adaptation methods. All in all, our proposed combined algorithm, GVBR, reduces the paly failure time from 26s to 1s, with an improvement of more than 96%.

References

[1] http://home.ifi.uio.no/paalh/dataset/hsdpa-tcp-logs/.
[2] https://media.xiph.org/video/derf/.
[3] https://obsproject.com/.
[4] https://www.fcc.gov/measuring-broadband-america.
[5] http://www.videolan.org/developers/x264.html.
[6] L. De Cicco, S. Mascolo, and V. Palmisano. Feedback control for adaptive live video streaming. In *MMSys*, 2011.
[7] J. Huang, C. Krasic, and J. Walpole. Adaptive live video streaming by priority drop. In *AVSS*, 2003.
[8] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. *SIGCOMM CCR*, 2015.
[9] J. Jiang, V. Sekar, and H. Zhang. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. *TON*, 2014.
[10] C. Krasic, J. Walpole, and W.-c. Feng. Quality-adaptive media streaming by priority drop. In *NOSSDAV*, 2003.
[11] Z. Lai, Y. C. Hu, Y. Cui, L. Sun, and N. Dai. Furion: Engineering high-quality immersive virtual reality on today's mobile devices. In *Mobicom*, 2017.
[12] B. Seo, W. Cui, and R. Zimmermann. An experimental study of video uploading from mobile devices with http streaming. In *MMSys*, 2012.
[13] R. Shea, J. Liu, E. C.-H. Ngai, and Y. Cui. Cloud gaming: architecture and performance. *IEEE Network*, 2013.
[14] M. Siekkinen, E. Masala, and T. Kämäräinen. A first look at quality of mobile live streaming experience: the case of periscope. In *IMC*, 2016.
[15] S. K. Singh, H. W. Leong, and S. N. Chakravarty. A dynamic-priority based approach to streaming video over cellular network. In *ICCCN*, 2004.
[16] J. Song, Y. Cui, Z. Li, Y. Bao, L. Zhang, and Y. Zhang. Edash: Energy-aware qoe optimization for adaptive video delivery over lte networks. In *ICCCN*, 2016.
[17] B. Wang, X. Zhang, G. Wang, H. Zheng, and B. Y. Zhao. Anatomy of a personalized livestreaming system. In *IMC*, 2016.
[18] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 2004.
[19] S. Wilk, R. Zimmermann, and W. Effelsberg. Leveraging transitions for the upload of user-generated mobile video. In *Proceedings of the 8th International Workshop on Mobile Video*, 2016.
[20] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli. A control-theoretic approach for dynamic adaptive video streaming over http. *SIGCOMM CCR*, 2015.
[21] C. Zhang and J. Liu. On crowdsourced interactive live streaming: a twitch. tv-based measurement study. In *NOSSDAV*, 2015.