VSync: Cloud Based Video Streaming Service for Mobile Devices

Eilwoo Baik, Amit Pande, Zizhan Zheng and Prasant Mohapatra
Department of Computer Science
Univerisity of California, Davis
Email: {ebaik, pande, cszheng, pmohapatra}@ucdavis.edu

***Abstract*—Synchronizing videos over file-hosting services on personal cloud such as Dropbox, Box or Onedrive leads to wastage in bandwidth and storage, which can be critical, while using mobile devices. Users can alternatively download the video on-the-go, but that leads to high latency, depending on network bandwidth and video file size. In contrast, adaptive video streaming allows near-real-time viewing by streaming the best possible quality in a given network condition. This feature is achieved by keeping multiple versions of video in cloud, leading to additional costs in cloud storage. Moreover, current solutions can only support a small set of bitrates, leading to abrupt switches in video resolution especially when the network condition is unstable, as often experienced by mobile users. This paper introduces `Vsync`, a framework for cloud based video synchronization for mobile devices. A video content is streamed using a cloud-based real-time transcoding and transmission framework to provide smooth video quality. Built over prediction models for video transcoding sessions and a QoE based adaptive video streaming protocol, `Vsync` is able to obtain the improvements of $37 \sim 80\%$ than other compared schemes. The dataset and evaluation was done on a pool of 220K video clips.**

Index terms - Mobile Video Streaming, Transcoding, MPEG-DASH, Cloud Service, Mobile Video Quality

I. Introduction

The ability to backup and store photos, videos, documents and personal content in the cloud file-hosting services, and automatically synchronize them across multiple devices have radically changed the ways people use personal devices. The rapidly increasing growths in the adoption of multiple devices by users along with the growing Internet access is a major factor in the transition to cloud-based services that can provide ubiquitous access to content and applications through any device at any location. With the help of cloud services, people can simply store various types of files, connect easily them from different multiple devices and also share them to others over Internet. Beyond storage, recently the cloud service providers have started offering availability, multi-platform support, security, app-integration, and more. According to reports [3], the industry will have more storage space in internal/private cloud service environments such as Box, Dropbox, Google Drive, and One-Drive than in traditional environments by the end of 2016. The global cloud storage traffic is expected to grow from about 2 exabytes (EB) annually in 2014 to 19.3 EB by 2018 at a CAGR of 56%.

Cloud based file storage services have been popular, because it allows users to view and share content over their mobile devices on-the-go. However, video content is still not that much popular in cloud stroage. Video content increases the demand for storage than any other content due to its larger size. This naturally leads to intensive network bandwidth consumption to synchronize the heavy video files in all devices connected to cloud service. However, *the mobile devices are ill-suited for syncing heavy data and files like video contents due to the limitations such as battery, low/unstable network bandwidth, local disk space and other resource shortages.* Best selling iPhone 5s/6has usual capacity of 16 GB memory, which can be lower for lower-end smartphones. However, 1 hour of HD movie may easily consume 2 GB of memory [4]. Moreover, it may take hours to synchronize over a typical broadband connection [4] consuming lots of bandwidth and battery resources. Moreover, synchronizing by downloading the video contents as a regular file does not allow receivers to play immediately. If the video file is of high quality or large duration, the start time (initial buffering time) will be very large, leading to irritation in end-users. A UDP-based realtime streaming service can be instead used, to reduce initial buffering time. However, UDP-based streaming suffers from video artifacts and poor video quality in low bandwidth scenarios.

Dynamic adaptive streaming over HTTP (referred to as MPEG-DASH or simply DASH) is getting popular as it supports different bitrates to adapt with given network bandwidth [15]. DASH-based schemes also allow near-real-time viewing of videos. However, such TCP/HTTP based adaptive streaming schemes require storing multiple resolution versions of videos in the server side, leading to more space and additional costs in the case of a cloud storage service. Another limitation of using DASH-based schemes is that it can only support a small set of bitrates and resolutions, leading to abrupt switches in video resolution especially when the network bandwidth is unstable, as often experienced by mobile devices. Absence of the consideration of user quality of experience (QoE) in adaptive bitrates changes gives a lower Mean Opinion Score (MOS) [6].

In this work, we introduce the notion of `Vsync`, a framework for cloud based video synchronization for mobile devices. Unlike general DASH-based schemes, which uses multiple video copies on the server, `Vsync` proposes to use real-time video transcoding scheme in allocation with video streaming service, to support a large number of bitrates and provide best possible video quality of experience (QoE) to end users. The proposed real-time transcoder is able to supply the best available video QoE, as well as a smooth transition in video resolution to improve visual experience. Unlike DASH-based service providers such as Youtube and Netflix, which use caches and multiple versions of video to optimize the video delivery [16], we use a single copy of video in server (at highest quality) to save memory resources in personal-cloud. This is justified for personal cloud, since the video will be viewed by a few users for a few times only. One big hurdle

in developing a video streaming scheme based on real-time transcoding with quality guarantees is variable transcoding time (TT) and transcoding size (TS), in addition to uncertainty of network bandwidth. To overcome it, we build predictive models for TT and TS using machine learning approach over features generated using codec parameters. We further develop a data-driven mobile-video QoE model to estimate video QoE delivered in mobile device under given video resolution, and develop an adaptive video streaming algorithm that provides a performance guarantee on long-term average QoE, under reasonable assumptions on transcoding and network dynamics. `Vsync` is able to obtain higher resolutions, less rebuffering events and higher QoE. The dataset and evaluation was done on a pool of 220K video clips. The main contributions of this work are summarized below:

- `VSync` : The paper proposes a cloud-based realtime transcoding and video delivery framework based on DASH protocol () to provide on-demand and smooth video streaming service to mobile users.
- A predictive model is developed to estimate video transcoding time ($\overline{TT}$) and transmission size ($\overline{TS}$) from video and codec parameters with 96% and 99% correlation accuracy with 5-15% error.
- A predictive model is developed to estimate video QoE in mobile devices for the given video playback resolution. It has 81% correlation to actual subjective ratings.
- An adaptive video-streaming algorithm is proposed, which provides a performance guarantee on long-term average QoE and outperforms DASH by [37.54 $\sim$ 67.359]% in various experiment conditions.
- `VSync` framework can run conveniently in smartphones. File-hosting services such as Dropbox and OneDrive can use VSync to optimize user experience for video service.

The paper is organized as follows. Section II discusses the motivation and related works on video streaming and transcoding. Section III provides an overview of the `VSync` system and the dataset we use. The transcoding model and our machine learning approach for estimating transcoding time and transcoded size are detailed in IV. The adaptive video streaming algorithm is presented in Section V. In Section VI, we evaluate the performance of `VSync` with real experiment and compare it with DASH implementation [1]. We conclude the paper in Section VII.

II. Background

In this section, we provide an overview of existing works on video streaming services and video transcoding services, and highlight the deficiencies of current video delivery services in the context of mobile video streaming.

A. Video Streaming Services

We first review the possible approaches to share uploaded video content to various mobile devices. The large file size of videos discourages us to synchronize them all-the-time like other file contents.

Conventional Downloading: Once a video content is downloaded in the local space, a user can play and enjoy with full playback-controls like FF (Fast Forward), RW (Rewind) or skip. However, any portion of video cannot be accessed till fully downloaded in the local disk, leading to start delay. **TCP-based Progressive Download (PD)** splits video file into multiple segments which are progressively downloaded. It reduces the start time, but does not support multiple quality levels of video. **UDP based streaming schemes** over RTP/RTCP is handy to transmit content in real-time and support large video file size. However, its lossy nature is barely acceptable for cases of high quality video contents services.Unlike UDP-based delivery, **TCP-based Adaptive streaming** schemes can offer reliable transmission with the recovery mechanism of TCP. DASH also allows adapting video quality to network conditions by keeping multiple copies of video in cloud and streaming the best possible quality in a given network condition [15], [29]. However, keeping multiple copies leads to extra storage overhead, which may not be economically viable for personal cloud where a video is often viewed by a few users for a few times. Moreover, this cost cannot be easily shifted to the end users. There have also been recent works on using video caching in wireless networks [13], [5]. However, although video caching is an attractive solution for video-on-demand services, it is not as efficient for personal cloud where the hits per video is orders lower than the millions. Moreover, in our context, the videos are often private and caching them may not be appropriate or economically feasible.

Vsync: Vsync uses an adaptive scheme to stream best visual quality to the end user based on device support and network bandwidth. It uses a visual QoE model to maximize perceptual video quality at end user. It keeps high quality video source at server end and transcodes the video at real-time to the desired video quality, based on video content, transcoder and network bandwidth.

B. Video Transcoding

Some commercial cloud service vendors such as Amazon Web Services (AWS), Bitcodin and Spritdsp support transcoding service to allow a user to make different resolutions for uploaded video files. A user can enjoy video contents in different devices having different resolution sizes. For example, an iPhone 4 user with a small resolution size (640×960p) cannot play a video content of 1280×1080p, requiring downsizing resolution/bitrates. Zencoder provides a transcoding service with different pricing rules and also supports RackSpace Cloud files. AWS provides an Elastic Transcoder that runs video transcoding jobs using the Amazon Elastic Compute Cloud (Amazon EC2) and stores transcoded video content files in Amazon Simple Storage Service (Amazon S3). The mentioned service providers do not support realtime transcoding schemes on a user on-demand video service.

There are a few recent works on cloud-based dynamic scheduling of video transcoding [19], [18]. Some authors consider autmatic tuning of encoding cost [18] or parallelizing transcoding [30], [17] but they focus only on video uploads. *Most of them focus on transcoding uploaded mobile videos and not on real-time streaming*. In addition, none of these work consider the joint transcoding and video delivery service with network dynamics taken into account. For example - H. Ma *et al*. [19] propose dynamic scheduling for video transcoding, but consider only video upload.

III. Vsync overview

The previous section outlined some deficiencies of existing video delivery services for video delivery to mobile devices from personal cloud. In this section, we give an outline of the `Vsync` framework, and discuss how it achieves the goals of (a) high user Quality of Experience (QoE), and (b) efficient

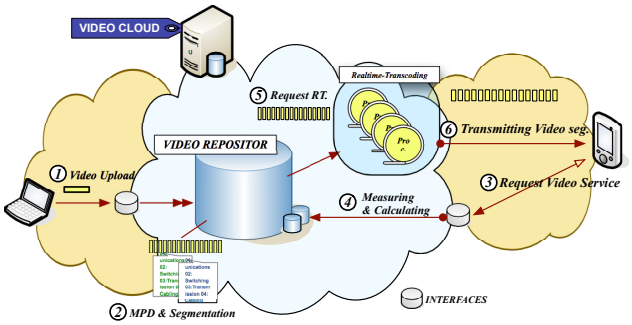


Fig. 1. Vsync framework.

space utilization in cloud and mobile device. Figure 1 gives an overview of Vsync framework. (1) User uploads video to the cloud. (2) Media Presentation Descriptions (MPDs) and Segmentation takes place at this stage. (3) Whenever a user requests for video in mobile device, a connection is established with video streaming service/repository in cloud. (4) The server estimates the available Network bandwidth and obtains current buffer-occupancy. This is done iteratively for each segment with the subsequent steps: (5) The video streaming service estimates the appropriate video resolution for the next segment based on network bandwidth, buffer-occupancy as well as estimates of transcoding time and transcoded file size, in order to maximize user's QoE. (6) Video is transcoded and transmitted to the mobile device.

Vsync achieves high QoE through the real-time video transcoding scheme. The real-time transcoding provides best available video QoE, as well as a smooth transition in video resolution to improve visual experience. This is enabled by an accurate estimation of the transcoding time and transcoded size of video segments via machine learning (Sections IV), a data-driven mobile video QoE model, and an adaptive scheduling algorithm for joint transcoding and transmission under network dynamics (Section V).

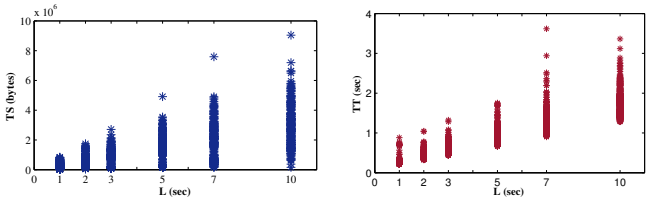
Dataset: We collected over 220K video clips (of 6K movie contents released in last 3 years) from Youtube. The video contents include all genres: adventure, drama, sports, action, fantasy, game, animation, music video, movie trailer, education, documentary and romance. The downloaded video clips have resolution of 1080p and are reproduced with other 4 types of resolutions of 240p, 360p, 480p and 720p by using tools of ffmpeg and Sony Vegas pro 12 trial version. We used fixed audio bitrate of 64kbps. The details of this dataset are given in Table I

IV. Vsync Transcoding Model

Vsync uses real-time transcoding so that a content server can transcode video segments according to availability of network resources and user's device capacity to maximize QoE. The length of each segment is limited to 10 seconds (similar to DASH standard). However, unlike DASH-based scheme which selects and streams pre-encoded and pre-uploaded copy of videos, Vsync uses an algorithm (discussed in next subsections) to decide the best resolution and bit-rate for the video segment. These parameters are then passed on to a video transcoder along with original high quality video content. The transcoder transcodes the video to desired resolution and the video segment is delivered to the mobile device. The pearson correlation between TT and L is high

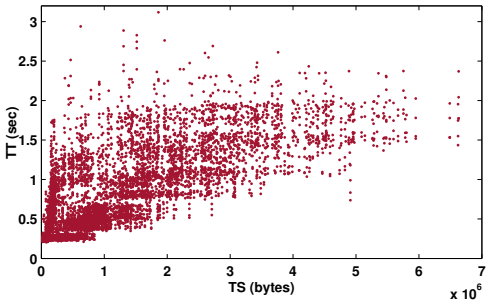
TABLE I
Vsync Video Dataset

Total # of Video Clips	220154
Total # of Video Contents	6188
Display Aspect-Ratio	16:9, 5:4, 4:3
Resolution Range (H)	[168 $\sim$ 1080p]
GOP	[4 $\sim$ 180]
Configuration # of B	[1, 5]
Configuration # of P	[1, 3, 5, 7, 8, 14, 19, 25, 29]
Frame rate	23.976, 24, 25, 29.97, 30
Audio bitrate (bps)	64
Segment Duration (sec)	[1, 2, 3, 5, 7, 10]
Segment size (bytes)	[21,921 $\sim$ 9,040,536]
Color Depth	8, 16bit
Encoder (codec)	mpeg4, H.264
Codec profile	MP4, H.264(base and High profile)
Container	avi, mp4



(a) Plot showing variation of Transcoded-segment Size (TS) with duration of TS (L).

(b) Plot showing variation of Transcoded-segment Size (TS) with duration of TS (L).



(c) Non-linear relationship between TT and input value

Fig. 2. Relation between transcoded size and transcoding time

(0.8721) but a linear regression gives high error ($> 60\%$). The relationship between TT and TS is plotted in Figure 2(c). A small correlation (0.5027) was found between the two. These results are obtained over the large pool of our initial dataset, and are thus representative of video-data streamed over Internet. To understand the confounding factor in this case, we tried to study the role of video codec parameters in TT and TS.

To effectively decide the best resolution for the video segment, the streaming algorithm needs estimate of (1) Transcoding Time (TT) and (2) Transcoded file Size (TS). Accurate prediction of TS of a segment is important in order to calculate transmission time in the given network bandwidth to meet the deadline for playback of a receiver. However, TT and TS vary with different videos as they depend on the content and compression codec parameters of the video. For example, slow motion video segment has much smaller size than the faster one. To proceed forward, we need to build a prediction model, which can estimate TT and TS based on available codec and content parameters.

Intuitively, one would assume that TS and TT depend on

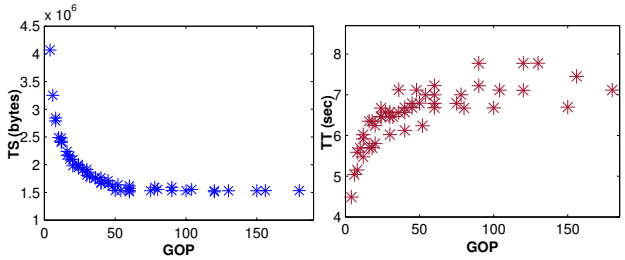


Fig. 3. Plot of TS (Left) and TT (Right) with increase in GOP size (Example segments of movie $Blackfish$)

the duration of trancoded segment. The larger the segment duration, the larger the values for TT and TS. To validate this assumption, we conducted experiments using VLC streaming player (transcoder) and study the plot between TS and duration. Based on the standard ISO/IEC 23009-1of MPEG DASH, the media segment (duration) can vary (2∼10sec for short, 10 or longer for long duration). Segments of duration 1,2,3,5,7 and 10 seconds were considered, because these are the popular range of segment sizes for adaptive streaming algorithms. Figure 2(a) shows the results. There is a positive correlation between duration and TS (pearson correlation coefficient was 0.5961). However, the relationship is not linear. A similar trend was obtained when we plotted TT versus duration. These results are plotted in Figure 2(b). In video coding, GOP refers to Group of Picture frames (successive) of a video which are coded together. Thus, a larger GOP size would facilitate compression and reduce TS. A larger GOP size may increase complexity of predictive coding technique (such as MPEG) and increase TT. Our experiments validated our assumptions. Figure 3 shows the variation of TT and TS with increase in GOP size for a sample video. The same trend is true for other videos in our pool. There is an exponential decrease in TT with increase in GOP size while the vice-versa is true for TT. The plot of TT is not smooth, may be due to variation in GOP configuration (such as ratio of B to P frames : $\frac{N_B}{N_P}$).

To account for these and other confounding factors, we build machine-learning based models to predict TT and TS. The confounding factors and L values are used as input features to the algorithm. Related codec parameters are listed in Table II.

A. Estimating Transcoding Time ($\overline{TT}$)

TT doesn't have a linear dependence on TS or TS duration or other video codec parameters. It varies with different video properties such as color depth, fps, GOP size and B and P frame configuration. TT for i^{th} segment (S_i) is estimated by the following relation:

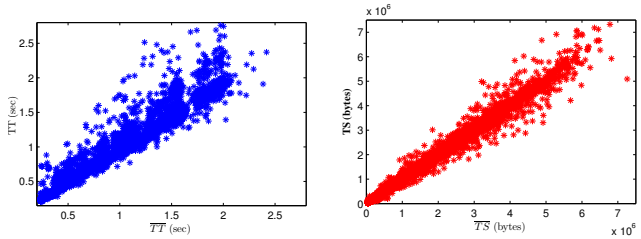
$$\overline{TT}(S_i) \leftarrow \Phi(v_1^{S_i}, v_2^{S_i}, v_3^{S_i} .....v_n^{S_i}) \quad (1)$$

where Φ represents the machine-learning model and $v_i^{S_i}$ represents feature vectors. **Feature Vectors**: The feature vectors $v_i^{S_i}$ for segment S_i are enumerated below:

1) L : Playtime of a segment.
2) CP : Codec profile (MP4, H.264/264$^+$).
3) fps : Video frame rate; (frame per second)
4) N_B, N_P : Number of video frame type B and P
5) N_{gop} : Size of GOP; $N_{gop} \leftarrow (N_B + N_P + 1)$
6) R_s : Resolution size; (H)
7) sz : Segment size

TABLE II
VIDEO CODEC PARAMETERS

Symbol	Meaning
H, W	Frame height and width
fps	Frame rate of video
R_a, R_v	Audio bitrate, Video bitrate
O_{vh}	Overhead ratio of codec
r	Frame Ratio $r = H/W$
R_{pix}	Pixel rate, $R_{pix} = H \times W \times fps$
R_b	Video bitrate, $R_b = \frac{R_{pix}}{CP}$
BPP	Bits per pixel, $BPP = \frac{R_b}{R_{pix}} \times 10^3$
C_d	Color depth, $C_d = \log_2(BPP \times 10^3)$
R_{tb}	Total bit-rate $R_{tb} = (R_b + R_a) \times (1 + O_{vh})$
sz_i	Size of i^{th} video segment, $sz_(i) = R_{tb}(i) \times L$
CP	Codec Profile, $CP = \begin{cases} 8000 & \text{for H264}^+ \\ 7000 & \text{for H264} \\ 5000 & \text{for MP4} \end{cases}$



(a) Actual and predicted TT values by Bagging model (Φ)

(b) Actual and predicted TS values by Proposed Model (Π)

Fig. 4. Comparisons between actual and predicted values of TT and TS

Regression Tree: Our model is built using Bootstrap aggregation (Bagging) [9], an ensemble machine learning technique built over reduced error pruning decision trees as the underlying regression model [14]. The bagging technique is an ensemble meta-algorithm to improve the stability and accuracy in statistical regression obtained by decision tree. The decision tree is based on information-theoretic criterion for selecting the nodes, and it auto-selects the most important feature vectors $v_i^{s_i}$ at top of tree. Once the tree is built, reduced error pruning is used, where each node, beginning with the leaves, is replaced with its most popular class. We divide the data for the model into n = 10 folds, where, n-1 folds are for supervised learning and one fold is used to test the model for errors. The errors obtained in a fold are added to the weights of nodes of next fold in the training set. Ten-fold cross validation was used to evaluate the model in order to ensure that the model was tested on data that it had not seen while training, to minimize chance for over-fitting. Statistical analysis was performed using Weka 3.6.10 [28] and Matlab R2013a (Ver 8.1.0.604) software.

Our model shows high correlation between $\overline{TT}$ and actual TT values (96%) with low error (see Table III). Figure 4(a) shows the distribution of predicted and actual TT values using our model.

B. Estimating Transcoded Size ($\overline{TS}$)

Like TT, TS also has dependence on multiple features. Hence, we model it using a bagging regression tree.

$$\overline{TS(S_i)} \leftarrow \Pi(v_1^{S_i}, v_2^{S_i}, v_3^{S_i} .....v_n^{S_i}) \quad (2)$$

where Π represents the machine-learning model and $v_i^{S_i}$ represents feature vectors.

Feature Vectors: The feature vectors $v_i^{S_i}$ for segment S_i are same as those for $\Phi()$ with the following additions:

1) C_d : Color depth
2) $\overline{TT(i)}$: Estimated Transcoding Time (from $\Phi()$)
3) $TBS(i)$: Transcoded bytes/second; $TBS(i) = \frac{sz}{\overline{TT(i)}}$

Regression Tree: A bagging regression tree based on reduced error pruning was built with 10-fold cross-validation approach. The proposed model shows 99.7% correlation with actual with a small relative absolute error (4.3%)on our dataset comprising over 220K video clips. The results are shown in Table III and Figure 4(b). Accurate estimation of $\overline{TS}$ and $\overline{TT}$ is critical to estimation of network delay.

TABLE III
BAGGING REGRESSION TREE MODEL FOR $\overline{TT}$ AND $\overline{TS}$

Type	$\overline{TT}$	$\overline{TS}$
Correlation Coefficient	**0.9614**	**0.9966**
Relative absolute error	15.018%	4.288%
Root Relative Squred Error	27.5072%	8.179%

V. VSYNC STREAMING SERVICE

A. QoE Metric

We design a QoE metric for TCP-based video. QoE for TCP-based delivery is dependent on (a) video resolution [12] and (b) freezing events [10]. In our streaming model (discussed in next subsection), buffer occupancy maximization or freezing minimization is considered as an independent goal. Thus, the QoE model considers video resolution and video content information as inputs. Video content information is modeled using Temporal Variation Metric of received video (TVM_k), which has been shown to efficiently model temporal information in video content across varied network conditions [10]. Formally, the QoE value for the i-th segment with resolution H_i and temporal information TVM_i^r is defined as follows.

$$\overline{QoE_i} = \alpha \cdot TVI_i^r + \beta \cdot H_i + \gamma \quad (3)$$

$$H_i \in [240p, 1080p] \quad (4)$$

Dataset: A subset of 2200 HD video clips form our dataset for QoE experiments. It was collected from real-world movie trailers collected over Youtube. Each video clip is reproduced for following resolutions: 240p/360p/480p/720p/1080p. This huge data set was evaluated by 183 people aged from 16 to 62 to give subjective ratings depending on different display resolutions in mobile devices. Subjective ratings vary from 1 (poor quality) to 5 (best quality). Our test was performed according to the ITU single-stimulus (SS) method [23]. The standard videos with the five different scores were shown and trained to the viewer at the beginning of the test. During the test, only the videos to be scored were shown without any display of the standard videos.

Results: Using linear regression, we obtained the following values: $\alpha = 6.5789$, $\beta = 0.002$ and $\gamma = -4.5598$. The model shows 82% correlation to actual scores. This model is chosen as a representative QoE model for video streaming service.

B. Streaming Modeling

We consider an extension of the adaptive streaming models in [15], [29] by incorporating real-time transcoding. *For simplicity, we ignore Start-Segments in this and the next section, which can be easily incorporated into our algorithm.* We formulate the QoE maximization problem from a single user's perspective. Extension to multiple users competing for transcoding and transmission is part of our future work.

TABLE IV
PERFORMANCE OF LINEAR REGRESSION MODEL FOR QOE

Type	Value
Corr. Coef.	0.8206
Rel. Abs. Error	55.253%
Root Rel. Abs. Error	57.023%

TABLE V
NOTATIONS IN VIDEO STREAMING MODEL

Symbol	Meaning
N	number of segments
L	playtime of a segment
T	time horizon for downloading all the N segments
C_t	network bandwidth at time t
$\widehat{C}_t$	average network bandwidth up to t
$\overline{C}$	long-term average network bandwidth
B_t	occupancy of playback buffer (in seconds) at time t
$B_{\max}$	size of playback buffer (in seconds)
R_i	bitrate of i-th segment

A video stream is modeled as a sequence of N video segments (or chunks), each containing L seconds of video. For each segment i, the content serve picks a bitrate R_i from a set $\mathcal{R} = [R_{\min}, R_{\max}]$. The segment is then transcoded according to the bitrate chosen and transmitted to the user. For simplicity, we assume that any bitrate between $R_{\min}$ and $R_{\max}$ is supported, which is a reasonable assumption since video segments are generated by the content server at real-time. At the user side, video segments are downloaded into a playback buffer. Let $B(t) \in [0, B_{\max}]$ denote the buffer occupancy at time t, defined as the total play time (in seconds) of unviewed video in the buffer. $B_{\max}$ is assumed to be relatively large (but finite), which is reasonable for DASH style downloading based streaming service. Table V summarizes the notations used.

To maximize the efficiency, the transcoding and transmission of video segments are parallelized. Transcoded segments are first saved in a queue, and then transmitted to the user in a FIFO order. We assume that the server decides the bitrate of segment i and starts transcoding the segment once it finishes transcoding segment $k-1$. [1] Let r_i denote the time at which the server starts to transcode the i-th segment. Let s_i and t_i denote the time at which the server starts and finishes to transmit the i-th segment, respectively. We assume that the server delays the transmission when the playback buffer at the client is full. It follows that $s_i \geq \max\{r_{i+1}, t_{i-1}\}$. That is, the server can start transmitting segment i only if it has finished transcoding segment i and also finished transmitting segment $i-1$. Without loss of generality, we assume $r_1 = 0$. Define $T \triangleq t_N$ to be the time horizon for downloading all the N segments. For any segment with bitrate R, let $TT(R)$ and $TS(R)$

[1] If the transcoding time is typically less than the transmission time of a segment, transcoding can be delayed so that decision making can be based on more recent information. For simplicity, we will not consider this in our algorithm.

denote its real transcoding time and transmitted segment size, respectively, which are unknown before transcoding, but can be estimated. We assume that $\mu(R) \triangleq \frac{TS(R)}{LR} \in [\mu_1, \mu_2]$ for any $R \in [R_{\min}, R_{\max}]$, where μ_1 and μ_2 can be estimated for a given video stream (see Section VI). Let C_t denote the bandwidth perceived by the player at time t. We have

$$TS(R_i) = \int_{s_i}^{t_i} C_t dt \tag{5}$$

$$t_i - s_i = \frac{TS(R_i)}{C_i} \tag{6}$$

where $C_i = \frac{1}{t_i - s_i} \int_{s_i}^{t_i} C_t dt$ is the average downloading speed over the period $[s_i, t_i]$. We further have the following dynamics for the playback buffer, where $(x)^+ \triangleq \max(x, 0)$:

$$B(t_i) = \left[B(s_i) - \frac{TS(R_i)}{C_i} \right]^+ + L \tag{7}$$

$$B(s_{i+1}) = [B(t_i) - (s_{i+1} - t_i)]^+ \tag{8}$$

Our objective is to design an online scheme at the server side that chooses the bitrate for each segment to maximize the user-perceived average quality-of-experience (QoE), i.e., $\frac{1}{N}\sum_{i=1}^{N} QoE_i$, with the additional objective of minimizing the number of rebuffering. Since $R_i = W \times H_i \times fps/CP$, we have $QoE_i = \alpha TVM_i^r + \beta H_i + \gamma = \alpha TVM_i^r + \frac{\beta \times CP}{W \times fps} R_i + \gamma$. We assume that $\frac{\beta \times CP}{W \times fps}$ is a constant for a given video stream. Then the QoE value is a linear function of the bitrate with a constant coefficient.

The main challenge of the problem is due to the uncertainty of C_t. We make a mild assumption on C_t in this paper. Let $\overline{C} \triangleq \lim_{T\to\infty} \frac{1}{T}\int_0^T C_t dt$ denote the long-term average bandwidth perceived by the player. We assume that the limit exists and $\overline{C}$ is finite. Define $\rho(R) \triangleq \frac{TT(R)}{TS(R)/\overline{C}}$, and $\rho \triangleq \max\{\max_{R\in\mathcal{R}} \rho(R), 1\}$. That is, ρ is defined as the worst-case ratio between the transcoding time and transmission time of a segment (under $\overline{C}$) if this ratio is larger than 1; otherwise, ρ is set to 1. Intuitively, $\overline{C}/\rho$ can be viewed as the "effective" average network bandwidth taking into account the overhead due to transcoding. We observe that $\rho(R)$ is upper bounded by 1 in our experiment setting (see Section VI); hence transcoding does not introduce extra overhead. We consider a general ρ so that our algorithm applies to any transcoding and networking conditions.

C. Adaptive Streaming Algorithm

In this section, we propose an online algorithm that achieves a close to optimal performance with a low rebuffering probability, adapting the framework in [25], [11]. Since the QoE value is a linear function in bitrate with a constant coefficient, we focus on maximizing the time average bitrate $\frac{1}{N}\sum_{i=1}^{N} R_i$. TVM_i^r is temporal information of original video, which remains constant for the segment in consideration. The performance bound that we obtain can be easily translated into a bound on QoE. We note that while focusing on average performance, our algorithm also ensures a smooth video quality when the buffer size is relatively large.

We first observe that, when $N \to \infty$ (or equivalently, $T \to \infty$), the maximum achievable average bitrates when there is no rebuffering is upper bounded by $\overline{C}/\mu_1$, even if C_t is known in advance for the entire time horizon T.

Theorem V.1. *For any sequence of $\{R_1, ..., R_i, ...\}$ where there is no rebuffering,* $\lim_{N\to\infty} \frac{1}{N}\sum_{i=1}^{N} R_i \le \overline{C}/\mu_1$.

Proof. Assuming there is no rebuffering, we must have $NL \ge T - t_1 + L$, since the video player can start to play the first segment after t_1, and can start to play the last segment after T. It follows that

$$\begin{aligned}\frac{1}{N}\sum_{i=1}^{N} R_i &\overset{(a)}{\le} \frac{\sum_{i=1}^{N}\int_{s_i}^{t_i} C_t dt}{\mu_1 NL} \\ &\overset{(b)}{\le} \frac{\sum_{i=1}^{N-1}\int_{s_i}^{s_{i+1}} C_t dt + \int_{s_N}^{t_N} C_t dt}{\mu_1 NL} \\ &= \frac{\int_0^T C_t dt}{\mu_1 NL} \le \frac{\int_0^T C_t dt}{\mu_1 (T - t_1 + L)}\end{aligned}$$

where (a) follows from (5) and the assumption that $\frac{TS(R_i)}{LR_i} \ge \mu_1$, and (b) follows from the fact that $s_{i+1} \ge t_i$. Therefore, $\lim_{N\to\infty} \frac{1}{N}\sum_{i=1}^{N} R_i \le \lim_{T\to\infty} \frac{\int_0^T C_t dt}{\mu_1(T - t_1 + L)} = \overline{C}/\mu_1$. □

Since the average bitrate is upper bounded by $R_{\max}$, we assume that $\overline{C} \le \mu_1 R_{\max}$ in the following discussion for simplicity. We also assume that the real time bandwidth is always sufficient to support the minimum bitrate, i.e., $C_t \ge \mu_2 R_{\min}$ for any t, which can be easily relaxed. According to the above theorem, the upper bound can be obtained if we can set $R_i = \overline{C}/\mu_1$ for $i \ge 1$. For a finite buffer size, however, this is not always possible due to fact that the buffer can occasionally be empty. Moreover, this strategy may increase the rebuffering probability and also reduce QoE.

Let $\widehat{C}_t = \frac{1}{t}\int_{\tau=0}^{t} C_\tau d\tau$ denote the average bandwidth up to time t. To achieve close to optimal bitrates while maintaining a very low rebuffering rate, our online algorithm adapts bitrates by taking into account both $\widehat{C}_t$ and $B(t$ (learned from the end users). For any segment i, R_i is chosen according to the following rules (see Algorithm 1): (1) R_i is bounded by $R_{\max}$; (2) If the current buffer has at least one segment, i.e., $B(t) \ge L$, $R_i = \frac{(1-\epsilon)}{\rho\mu_2}\widehat{C}_t$. Intuitively, R_i is chosen to match the "effective" bandwidth $\widehat{C}_t/(\rho\mu_2)$ with both the overhead from transcoding and the inaccuracy of estimated transcoding size taken into account. The extra factor $(1-\epsilon)$ is introduced to reduce the chance of rebuffering, where ϵ is a parameter that can be adjusted; (3) If the current buffer does not have one or more segments, then R_i is further reduced by a factor $B(t)/L$ to reduce the chance of rebuffering.

Algorithm 1 Adaptive Transcoding and Streaming

For $i = 1, 2, ...$

1: $t \leftarrow$ time finishing transcoding $(i-1)$-th segment;
2: $\widehat{C}_t \leftarrow \frac{1}{t}\int_{\tau=0}^{t} C_\tau d\tau$;
3: $R_i \leftarrow \min\{R_{\max}, \frac{(1-\epsilon)}{\rho\mu_2}\widehat{C}_t, \frac{B(t)}{L\rho\mu_2}\widehat{C}_t\}$

Note that due to the uncertainty of C_t, Algorithm 1 cannot avoid rebuffering surely. We first consider the case when the playback buffer is unbounded. The following theorems shows that the algorithm achieves a close to optimal average bitrate as t approaches to infinity.

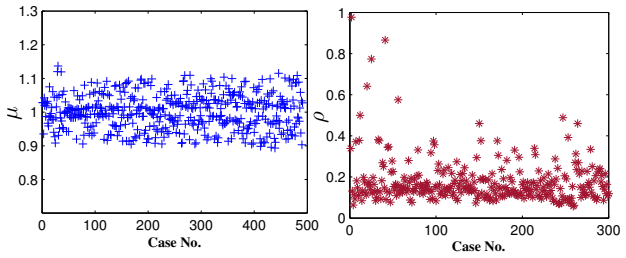


Fig. 6. Plot of μ and ρ values obtained in streaming sessions

Theorem V.2. *When the buffer size is infinite, Algorithm 1 achieves an expected bitrate of at least* $\frac{(1-\epsilon)^2}{\rho\mu_2}\overline{C}$ *as* $t \to \infty$*, with the probability of rebuffering approaches to 0.*

Proof. The theorem follows from a similar argument in [11]. We provide a proof sketch here for completeness. First, by the definition of $\overline{C}$, for any $\epsilon > 0$, there exists time t_1, such that $(1-\epsilon)\overline{C} \leq \widehat{C}_t \leq (1+\epsilon)\overline{C}$ for all $t > t_1$. Moreover, if we consider the playback buffer as a queue, then the service rate is upper bounded by $\frac{(1-\epsilon)}{\rho\mu_2}\widehat{C}_t$, while the average arrival rate is lower bounded by $\frac{1}{\rho\mu_2}\overline{C}$, which is strictly greater than the service rate for $t > t_1$. Therefore, the playback buffer will build up and $\lim_{t\to\infty} P\big(\frac{B(t)}{L} > (1-\epsilon)\big) = 1$. It follows that $\lim_{i\to\infty} P\big(R_i > \frac{(1-\epsilon)^2}{\rho\mu_2}\overline{C}\big) = 1$. A similar argument shows that once the buffer occupancy is large enough, and by the assumption that $C_t \geq \mu_2 R_{\min}$ for any t, the probability of rebuffering approaches to 0 as $t \to \infty$. □

On the other hand, when the buffer size is finite but large enough, the probability of rebuffering can be estimated by $\lim_{t\to\infty} P(B(t) = 0) \leq \exp(-\gamma^* B_{\max})$ where γ^* is a positive constant, following a similar argument in [11]. This result holds when the network bandwidth process C_t is *i.i.d.* or Markovian, and under some more general settings [22]. This implies that under these settings, by taking $B_{\max} = O(\ln \frac{1}{\delta})$ with δ relatively small, the rebuffering probability is bounded by δ almost surely, and furthermore, Algorithm 1 achieves an expected bitrate of at least $\frac{(1-\delta)(1-\epsilon)^2}{\rho\mu_2}\overline{C}$ almost surely. By taking $\epsilon = \delta$, this is no worse than a factor $\frac{(1-\delta)^3\mu_1}{\rho\mu_2}$ of the optimal according to Theorem V.1.

VI. Experiments & Implementation

In this section, we have implemented `VSync` over DASH [1], evaluate the performance and compare it with normal DASH implementation [1] (called as DASH). We show that `VSync` achieves a better QoE under various network conditions. In particular, we show that while DASH can occasionally provide higher video resolution, it also introduces more rebuffering and a lower overall QoE perceived by end users. In contrast, `VSync` maintains a low rebuffering rate and a smooth video quality.

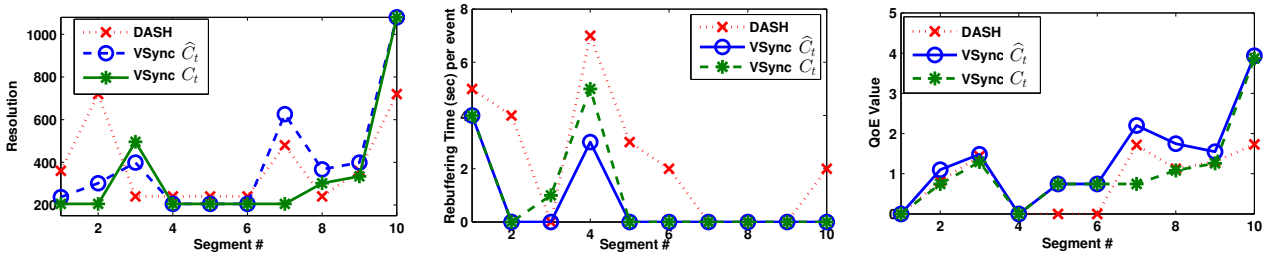
Experiment Setup : In order to evaluate `VSync` framework, we configure our testbed between a server that measures network conditions, does transcoding and other computations; and a client that receives the video segments and plays them during the given video streaming session. In our scenario, we assume that the server has a copy of the original video content. It is already split into multiple segments (10 sec each) with Media Presentation Description (MPD) files that elaborate the required video properties used in the service time. When the user double clicks on the video, the cloud server receives the service request and the current buffer occupancy from the client, and estimates the network bandwidth (BW) for the connection. Based on the video segment properties, the estimates of transcoding time ($\overline{TT}$) and Transcoded/Transmitted size ($\overline{TS}$) are obtained as a function of the resolution (H). These inputs are then provided to the streaming algorithm which decides the resolution for video that maximizes QoE. The playtime of each segment are fixed at 10sec in this work. A segment is processed in the transcoding-thread and enters the queue of transmitter of the server. The transcoding and transmission are done in an asynchronous pipeline, meaning that the transmission begins once the transcoder outputs a segment. In the process of transcoding, the segment can be split into smaller segments to be encoded by multiple processors (upto 3), and merged into a single segment to be sent. The local buffer size ($B_{\max}$) is bigger than the video content streamed. The segments arriving at the client are stored and automatically loaded into the the video player buffer for playback. A Free BSD-based iMAC (16GB, 3.2GHz Intel Core i7) is used as a server and connected to wired network connection. The receivers are tested in different wireless network conditions (IEEE 802.11n). The video contents are randomly chosen from the video pool. The codec/encoder/transcoder/decoder used in this implementation are implemented using FFmpeg [8] and VLC [24]. We evaluate two variants of `VSync`. The first version, called `VSync` $\widehat{C}_t$, uses the estimated average bandwidth $\widehat{C}_t$ as in Algorithm 1, while the second version, called `VSync` C_t, uses the instant bandwidth C_t instead of $\widehat{C}_t$ in Algorithm 1. Based on the standard of MPEG-DASH [26], [1] we also implemented DASH and compare it with both `VSync` versions for streaming performance. MPEG-DASH server has the video in 5 different resolutions: 240p, 360p, 480p, 720p and 1080p. A DASH client receives segments which is best-fit to current network bandwidth. They are uploaded to the server and dynamically selected during streaming. Thus, DASH has a competitive edge over `Vsync` that it saves time and complexity of accommodating real-time transcoding procedure. Our results are measured over 6∼20 measurements per streaming request and averaged out.

Evaluation Metric : `Vsync` aims at providing optimal user experience while watching videos in personal cloud. Thus, we evaluate its performance in terms of video QoE. To model QoE, we use raw measurements of video resolution delivered, total rebuffering and user QoE as evaluation metric.

We choose a recently published QoE model[2], called as visual acuity [6], to estimate QoE received by the end user. The visual acuity model was preferred over other visual quality models [7], [27], [21], [20] because (1) it models both freezing and resolution related impairments observed in TCP-based video delivery, (2) it gives a single score which matches closely to subjective ratings, (3) it is validated over large pool of 3000 Youtube video clips, and (4) it has superior performance than existing models.

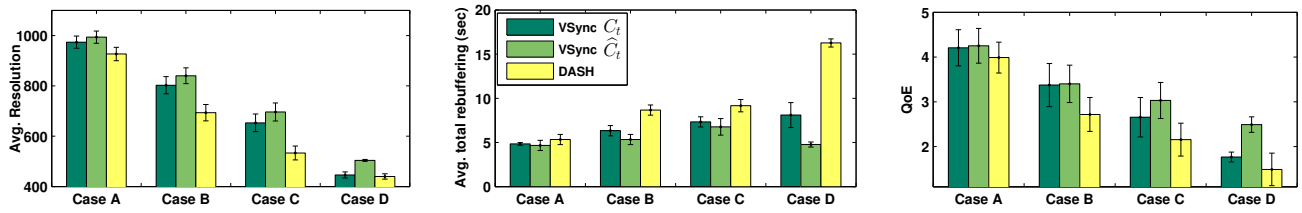
Validating the streaming model: In the last section, we made some assumptions regarding the streaming model. The

[2]This QoE model is different from the previous simplified QoE model, which was used by streaming service in the cloud. Unlike that model, this QoE model has higher accuracy (89%) and is based on receiver-side (mobile device) measurements during playback.



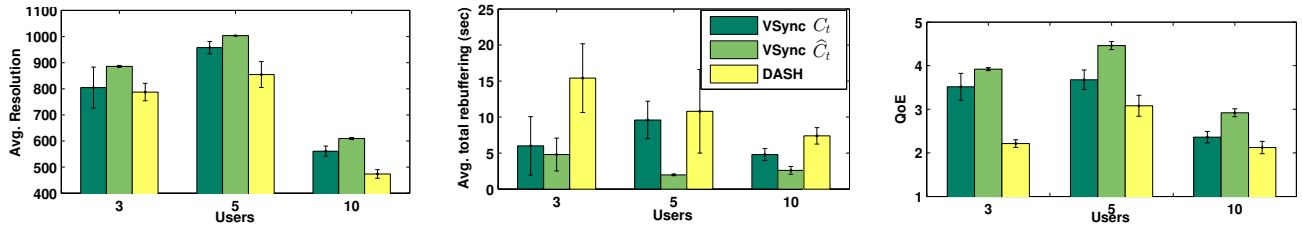
(a) Resolution Switches at each segment (b) Rebuffering event at each segment (c) QoE value at each segment

Fig. 5. Performance at the conditions of video length (100sec) and BW (avg : 607.37Kbps, $stdev$: 230.906)



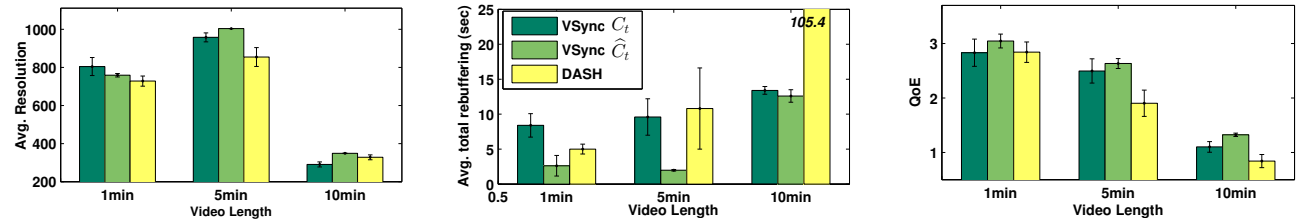
(a) Average Resolution (Bitrates) (b) Average total rebuffering (c) Average QoE value

Fig. 7. Performance at the video length (180sec) and network conditions of Case A [avg : 2.578, std : 0.9215], B [1.6941, 0.6035], C [1.29499, 0.466] and D [0.9477, 0.366] in Mbps



(a) Average Resolution (Bitrates) (b) Average total rebuffering (c) Average QoE value

Fig. 8. Performance at the different number of users and network conditions of 3users at [avg : 2.111342, std : 0.98601], 5users at [2.65828, 1.962224], 10users at [1.085517, 0.372023] in Mbps with video length (300sec)



(a) Average Resolution (Bitrates) (b) Average total rebuffering (c) Average QoE value

Fig. 9. Performance at the different video length and network conditions of 1min at [avg : 2.284369, std : 1.117054], 5min at [2.65828, 1.962224], 10min at [0.55478, 0.238453] in Mbps

first one was regarding the minimum and the maximum range of μ, which characterizes the uncertainty of transcoded segment size under different bitrates. The second assumption was regarding ρ, the worst-case ratio between the transcoding time and transmission time under the average network condition. In over 300+ live streaming sessions, we found that μ is largely restricted in the small range [0.89,1.13]. This indicates that the uncertainty of transcoding size has only a minor effect on the performance of our adaptive streaming algorithm. We further observe that ρ is largely small and is around 0.177 ± 0.105. The value of ρ approached the range [0.5,1.0] in some poor network conditions only. This implies that by parallelizing transcoding and transmission, transcoding does not introduce extra overhead in our experiment setting. Figure 6 gives a plot of the observations made about these variables in live streaming sessions.

Response to network conditions: Figure 5 gives a zoom-in segment-by-segment trace of video resolution, rebuffering and instantaneous QoE for a streaming session. DASH initializes with 360p resolution settings, but `Vsync` chooses lower resolution (205∼225p) so that it can transmit the first segment as soon as possible to minimize the startup latency. For subsequent 2-3 segments, there is a dip in network bandwidth. `Vsync` chooses shorter resolution than DASH and thus reduces rebuffering. In this short duration, `VSync` C_t has similar QoE as DASH while `Vsync` $\widehat{C}_t$ has slightly higher QoE.

Performance under varied network conditions : We tested the performance of `Vsync` under four network conditions. The average mobile network connection speed in 2014 was 1,683 kbps [2]. Thus, to make realistic assumptions we model four network bandwidth processes around this range.

The network bandwidth at any time instance follows a uniform distribution with mean avg and standard deviation std and is *i.i.d.* over time, where avg and std are given parameters. Case A has average speed of 2.6Mbps while cases B,C and D have average speeds of 65%, 55% and 35% of A's speed. Figure 7 gives a plot of video quality under different scenarios. Although all scheme perform well under good network conditions, we can see that DASH has a very high rebuffering for low network bandwidths, leading to poorer performance than `Vsync`. Moreover, we can observe that $\widehat{C}_t$ has lower rebuffering and slightly higher resolution than C_t. Figure 8 shows the performance of the schemes when multiple users and a single video streaming receiver are contending the network at receiver end. In this case, we adopt the network dynamics model suggested in [29]. At any time instance, there are n user accessing the network together where n follows the Poisson distribution (*i.i.d.* over time) with mean equal to 3, 5, or 10. The bandwidth for a single user then follows a uniform distribution with mean avg/n and standard deviation std/n, where avg and std are given parameters. We observe that DASH provides similar average resolution like the other two schemes, but has a higher rebuffering time for all the three cases. Vsync $\widehat{C}_t$ shows stable performance and lower rebuffering time than the other two, leading to higher QoE values. Vsync C_t has higher average resolution at 5 users than the case of 3 users, but there is no noticeable increments in QoE due to more rebuffering time.

Next, we test the performance for video clips of different duration. *We set these experiments in a real-world scenario* where the client and server are on different networks and connected via Internet and physically separated over 3 miles. Figure 9 shows the average results for video clips of duration 1, 5 and 10 minute each. For 10 minute clips we observed low network bandwidth, hence the QoE is lower. Although Vsync $\widehat{C}_t$ has lower avg. resolution than VSync C_t in Figure 9(a), Vsync $\widehat{C}_t$ achieves better performance in rebuffering. This is because Vsync C_t selected higher video bitrates according to instant network bandwidth, but it suffers from the fluctuating network conditions, leading to more rebuffering events. This finally brings lower QoE than Vsync $\widehat{C}_t$. In the Figure 9(a), VSync $\widehat{C}_t$ has net lower resolution than DASH for 10min. However, DASH has lower QoE values due to large amount of buffering (see Figure 9(b-c)).

VII. Conclusion

This work proposed `VSync`, a video streaming service for delivering video files in personal cloud. `VSync` uses real-time video transcoding on top of video streaming model to maximize user QoE. Experimental results indicate that the streaming model outperforms DASH , giving significant improvements in poor network conditions. Using Start-Segments can further improve the QoE (over 100% in poor network conditions). `VSync` is a promising protocol, for use by Personal Cloud providers such as Dropbox, OneDrive and Box to provide smooth video delivery to mobile users, without wasting loads of network bandwidth for pre-caching entire content.

References

[1] http://dashif.org/software/.

[2] *Cisco Visual Networking Index: Forecast and Methodology, 2013–2018*. Cisco, 2013.

[3] *Leading Cloud Computing, Big Data and Trusted IT managing http://www.emc.com/index.htm?fromGlobalSelector, last accessed March 2015*, 2015.

[4] *iTunes store: Download times will wary, https://support.apple.com/en-us/HT201587*, Accessed March, 2015.

[5] H. Ahlehagh and S. Dey. Video caching in radio access network: impact on delay and capacity. In *Proc. of WCNC*, pages 2276–2281. IEEE, 2012.

[6] E. Baik, A. Pande, C. Stover, and P. Mohapatra. Video acuity assessment in mobile devices. Proc. of INFOCOM, 2015.

[7] A. Balachandran, V. Sekar, A. Akella, S. Seshan, I. Stoica, and H. Zhang. Developing a predictive model of quality of experience for internet video. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 339–350. ACM, 2013.

[8] F. Bellard, M. Niedermayer, et al. Ffmpeg. *Internet: http://www. ffmpeg. org,[Dec. 27, 2012]*, 2007.

[9] L. Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[10] A. J. Chan, A. Pande, E. Baik, and P. Mohapatra. Temporal quality assessment for mobile videos. In *Proc. of Mobicom*, pages 221–232. ACM, 2012.

[11] S. Chen, P. Sinha, N. B. Shroff, and C. Joo. A simple asymptotically optimal joint energy allocation and routing scheme in rechargeable sensor networks. *IEEE/ACM Transactions on Networking*, 22(4):1325–1336, 2014.

[12] T. De Pessemier, K. De Moor, W. Joseph, L. De Marez, and L. Martens. Quantifying subjective quality evaluations for mobile video watching in a semi-living lab context. *IEEE Transactions on Broadcasting*, 58(4):580–589, 2012.

[13] N. Golrezaei, K. Shanmugam, A. G. Dimakis, A. F. Molisch, and G. Caire. Femtocaching: Wireless video content delivery through distributed caching helpers. In *Proc. of INFOCOM*, pages 1107–1115. IEEE, 2012.

[14] G. Holmes, A. Donkin, and I. H. Witten. Weka: A machine learning workbench. In *Proc. of the Second Australian and New Zealand Conference on Intelligent Information Systems*, pages 357–361. IEEE, 1994.

[15] 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. In *Proc. of SIGCOMM*, 2014.

[16] D. K. Krishnappa, M. Zink, C. Griwodz, and P. Halvorsen. Cache-centric video recommendation: an approach to improve the efficiency of youtube caches. In *Proc. of ACM Multimedia Systems Conference*, pages 261–270. ACM, 2013.

[17] P. Li, B. Veeravalli, and A. A. Kassim. Design and implementation of parallel video encoding strategies using divisible load analysis. *Circuits and Systems for Video Technology, IEEE Transactions on*, 15(9):1098–1112, 2005.

[18] X. Li, Y. Cui, and Y. Xue. Towards an automatic parameter-tuning framework for cost optimization on video encoding cloud. *International Journal of Digital Multimedia Broadcasting*, 2012.

[19] H. Ma, B. Seo, and R. Zimmermann. Dynamic scheduling on video transcoding for mpeg dash in the cloud environment. In *Proc. of ACM Multimedia Systems Conference*, pages 283–294. ACM, 2014.

[20] A. Mittal, A. Moorthy, and A. Bovik. No-reference image quality assessment in the spatial domain. 2012.

[21] A. Mittal, R. Soundararajan, and A. Bovik. Making a completely blindimage quality analyzer. 2012.

[22] P. W. Glynn and W. Whitt. Logarithmic asymptotics for steady-state tail probabilities in a single-server queue. *Advances in Applied Probability*, 10:131–156, 1994.

[23] I. Rec. Bt. 500-11, methodology for the subjective assessment of the quality of television pictures. *International Telecommunication Union*, 2002.

[24] V. S. Solutions. VLC media player, 2006.

[25] R. Srivastava and C. E. Koksal. Basic performance limits and tradeoffs in energy-harvesting sensor nodes with finite data and energy storage. *IEEE/ACM Transactions on Networking*, 21(4):1049–1062, 2013.

[26] T. Stockhammer, P. Fröjdh, I. Sodagar, and S. Rhyu. Information technology—mpeg systems technologies—part 6: Dynamic adaptive streaming over http (dash). *ISO/IEC, MPEG Draft International Standard*, 2011.

[27] Z. Wang, L. Lu, and A. Bovik. Video quality assessemnt based on structural distortion measurement. *Signal Processing: Image Communication*, 19(2):121–132, February 2004.

[28] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

[29] X. Yin, V. Sekar, and B. Sinopoli. Toward a principled framework to design dynamic adaptive streaming algorithms over http. In *Proc. of Hotnets*, 2014.

[30] M. Zaharia, D. Borthakur, J. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Job scheduling for multi-user mapreduce clusters, 2009.