Low-cost Video Transcoding at the Wireless Edge

Jongwon Yoon
Computer Science and Engineering
Hanyang University
Korea
Email: jongwon@hanyang.ac.kr

Peng Liu
Dept. of Computer Sciences
University of Wisconsin-Madison
Madison, WI 53706
Email: pengliu@cs.wisc.edu

Suman Banerjee
Dept. of Computer Sciences
University of Wisconsin-Madison
Madison, WI 53706
Email: suman@cs.wisc.edu

***Abstract*—With the proliferation of hand-held devices in recent years, wireless video streaming has become an extremely popular application. Internet video streaming to mobile devices, however, faces several challenges, such as unstable connections, long latency, and high jitter. Bitrate adaptive streaming and video transcoding are widely used to overcome such problems. However, there are still several shortcomings of these approaches. Bitrate adaptive streaming cannot provide fine-grained adaptation. Moreover, video transcoding is expensive and is hard to apply for live streaming.**

In this work, we propose to use a low-cost video transcoding solution running at the wireless edge, such as home WiFi Access Points (APs). Instead of having expensive server-based transcoding, we designed and implemented a real-time video transcoding solution on a low-cost hardware, Raspberry Pi. By running our transcoding solution at the wireless edge, it can provide more agile adaptation to sudden network dynamics and is able to incorporate clients' feedback quickly. Our transcoding solution is transparent, low-cost, and scalable. Thus it allows broad and quick deployment in home WiFi APs (and also in cellular base stations). The evaluations reveal that our system enhances the performance of video streaming compared with other bitrate adaptive solutions. It provides higher video bitrates (at least 2.1×) without causing video stall or rebuffering.

I. Introduction

HTTP traffic constitutes a significant percentage of network traffic, especially multimedia usage over HTTP accounts for an increasing portion of today's Internet traffic [29]. Worldwide smartphone sales overtook PCs in 2011 [17], and video streaming on handheld devices (e.g., smartphones, tablets) is one of the most popular applications. With the rapid increase in mobile device usage, the amount of traffic destined to mobile devices is also exponentially growing, and online video streaming constitutes the majority of this traffic [13], [1]. Given the popularity of mobile streaming, demand for high bandwidth in wireless mobile network continues to grow. The key to success for internet video streaming is to deliver high quality and provide interruption/distortion-free and continuous playback with an immediate start. In spite of the increasing demand for mobile video streaming, current wireless networks cannot keep up and provide satisfactory video streaming quality to mobile users. This is due to several factors; the wireless link is dynamic because of link fluctuation and user mobility, it provides limited capacity in multi-client deployments, and the wireless channels are prone to being unstable. These factors lead to undesirable outcomes such as user dissatisfaction.

Downsides of source-based adaptation. The challenges and limitations caused by wireless connections have been addressed by proposals for better video delivery approaches, such as adaptive bitrate streaming [5], scalable video coding (SVC) [7], and progressive downloading [6]. In the adaptive bitrate approach, the media server maintains multiple profiles of the same video, which are encoded in different bitrates and qualities before they are delivered to the end users. However, the wide variety of bitrates, codecs, and formats makes it difficult for the video service provider to predict and pre-process numerous videos in advance. It also imposes higher overhead on content providers and the delivery infrastructure such as media server and storage. For instance, Netflix maintains 120 different versions of each video to deal with different screen sizes and bandwidth requirements [39]. The limited selection of pre-processed bitrates can not provide an agile adaptation because channel instability frequently occurs with user mobility in wireless networks. Even worse, it is hard to pre-process live media sources like breaking news, popular sports events, and teleconferences in real-time. Other shortcomings of these schemes include the following: they are not dynamically configurable in sudden network changes, and some of the content may be wasted if it is not watched (e.g., progressive download used in YouTube or Hulu).

Video transcoding. Given the difficulties of pre-processing various formats of video streams and the challenges of fine-grained bitrate adaptation, video transcoding has emerged as an alternative technology for optimizing video data. Video transcoding encodes or converts the input stream to a pre-configured format (e.g., codec, resolution, bitrate) to overcome the diversity and compatibility issues presented by the multiple formats of video streams. It is commonly used in the area of mobile device content adaptation, where a target device does not support the format or has a limited storage capacity and computational resource that mandates a reduced file size. However, video transcoding is a very expensive process requiring high computational power and resources, thus it is usually performed in the media server in an off-line manner.

Several research papers published recently [8], [4] use video transcoding solutions in order to provide video streaming service to a wide range of devices. They propose a cloud transcoder to offload transcoding work in the cloud and make the reusing of transcoded videos possible by storing them in the cache. This approach requires storage and causes additional

978-1-5090-3322-5/16 $31.00 © 2016 IEEE
DOI 10.1109/SEC.2016.8

delay as video traffic is redirected from the cloud system. It does not support real-time video streaming because it is based on an off-line transcoding. In short, *video transcoding is a very effective way to optimize video data, however, it is expensive and is hard to apply for live streaming.*

TransPi. In this paper, we introduce a *cost-effective*, *real-time* video transcoding solution, TransPi, at the wireless edge where it can be agile to adapt to sudden network changes in order to provide a better streaming experience to the user. Instead of running on expensive transcoding process in the media server, we utilize a low-cost hardware, Raspberry Pi, for video transcoding running at the wireless edge, e.g., co-located or embedded in home WiFi AP. The Raspberry Pi is a cheap, credit card sized computer that includes a graphics processing unit (GPU) and a hardware decoder and encoder that is able to accelerate the video transcoding process. By running our transcoding solution at the wireless AP, it can provide more agile adaptation to sudden network dynamics and is able to quickly incorporate client feedback.

To achieve real-time transcoding on live streaming, we partition the input stream into multiple small segments and apply our transcoding solution on-the-fly. Then we provide the continuous transcoded video stream to the client in real-time with minimal delay. TransPi provides a seamless, flexible bitrate streaming service to the user, while taking into account the user's profile (e.g., downlink bandwidth). In contrast to other transcoding work and previously existing systems, we propose a storage-less, on-the-fly, and seamlessly adaptive transcoding solution. Our evaluations show TransPi provides a better streaming service to the user.

Wireless edge. In TransPi, we take a channel-aware approach for providing a better streaming service while incorporating instantaneous feedback from the client. Given the benefits of cloudlet architecture [27] for video streaming service, we deploy our transcoding solution at the wireless edge where it can monitor the client's profile from the nearest vantage point in order to maximize the QoS. Moreover, deploying video transcoding solution at the wireless edge provides more affordability to the end user and allows speedy deployment in existing systems (e.g., simply connecting the Raspberry Pi to the home WiFi AP). In this sense, TransPi is agile to network changes.

In a residential environment with dense WiFi deployments and severe interference (microwaves, Bluetooth devices, etc.), high variation in wireless network and sudden network changes play a bigger role and directly affect the user's experience of video streaming. Having higher network capacity is beneficial in some cases where many people share the bandwidth. However, it cannot directly help the cases where network is highly variable or unstable. Instead of high bandwidth, quick adaptation to the network changes can overcome such problem. Our video transcoding is deployed at the wireless edge where it quickly monitors the last hop changes. TransPi can instantaneously adapt the video bitrate according to the network changes so that it can provide interruption/distortion-free streaming service to the user.

Contributions. The most important contribution of our system is a practical implementation of real-time video streaming optimization based on a low-cost, hardware-assisted transcoding solution at the wireless edge (i.e., connected to the wireless AP). Our contributions in this work are multi-fold:

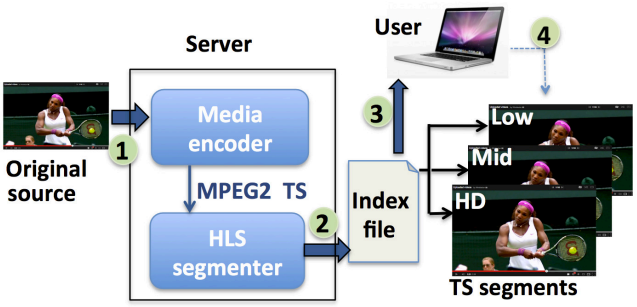
- We design and implement a cost-effective transcoding solution on low-cost hardware running at the wireless edge.
- Our solution is transparent, low-cost, and scalable, thus it allows wide and quick deployment in a home WiFi APs (or cellular base-station).
- TransPi partitions the input stream into small segments and applies transcoding solution on-the-fly, and hence it handles live streaming in real-time without interruptions.
- TransPi provides seamless bitrate switches with fine time granularity and efficiently utilizes available bandwidth by adapting bitrates of transcoded video to the network conditions.

The remainder of this paper is organized as follows. We introduce background of video streaming solutions widely used today in Section II. Section III motivates the transcoding solution in the cloudlet by comparing the performance to conventional bitrate adaptive streaming. Section IV describes the overview of TransPi, details of our transcoding solution and its implementation. We evaluate the performance of TransPi in Section V. We discuss related and future works in Section VI and VII, respectively. Section VIII concludes the paper.

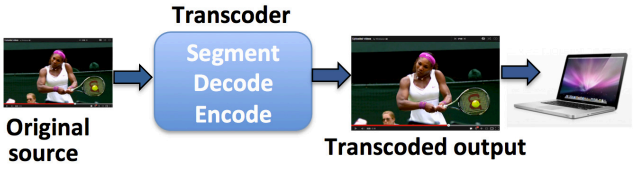
II. Adaptive Bitrate Streaming

Traditionally, video streaming solutions utilize either a UDP-based (RTP/RTSP and IP-layer multicast) or TCP-based (RTMP) approach while optimizing video data to promote high efficiency. HTTP-based solution is widely used these days due to the advantages of its extensive infrastructure, servers and caching. Many challenges, however, arise in delivering video data over HTTP; it is required to have various video formats to support diverse end users and maintain a stable connection between server and client, and the latency has to be carefully controlled for streaming live media. Especially in highly variable wireless channels, it is critical to adapt quickly to the channel conditions in order to provide the user with a good viewing experience. To overcome such challenges, many solutions utilize bitrate adaptive mechanism, HLS [9], MPEG-DASH [10], Adobe Dynamic Streaming[12] and Microsoft Smooth Streaming [11], however, they are not fully functional across different platforms which makes their usage very limited.

HTTP Live Streaming (HLS). HLS is an HTTP-based media streaming protocol implemented by Apple and supported by all Apple smart devices (iOS and Safari) and Android devices (Android v3.0 or higher). HLS is an open standard and provides great flexibility for implementation. A schematic view of HLS processes is depicted in Fig. 1(a). The media encoder downloads original video stream, encodes it into H.264 video format and generates an MPEG-2 Transport Stream (TS). Then, the stream segmenter takes the MPEG-2



(a) Adaptive bitrate streaming (HLS)



(b) Video transcoding example

Fig. 1. The bitrate adaptive streaming takes the original video and turns it into multiple version of various bitrates, whereas the video transcoding solution in TransPi generates a single stream.

TS and produces a series of equal-length files (TS segments) from it that are suitable for use in HLS. It also generates an index file (i.e., playlist) that contains a list of the media files. The client player reads the index file, requests the listed transcoded video data, and displays it without any pauses or gaps between segments. The client also switches between streams dynamically if the available bandwidth changes.

Dynamic Adaptive Streaming over HTTP (DASH). DASH uses HTTP web server infrastructure to deliver web content and allows many devices, such as Internet connected TV, TV set-top boxes and many mobile devices. Similarly to HLS, DASH works by breaking the overall stream into a sequence of small chunks. Before the client player (e.g., dash.js) begins downloading media segments, the client first requests the MPD, which contains the metadata for the various sub-streams that are available. As the stream is played, the client may select from a number of different alternate streams containing the same material encoded at a variety of bitrates, allowing the streaming session to adapt to the available network capacity.

Supporting both HLS and DASH formats in TransPi. In TransPi, we support the two most widely used streaming formats, HLS and DASH, while following their specifications, however TransPi does not employ their rate adaptation algorithms. In a nutshell, we use the video transcoding solution to generate video data whose bitrate spontaneously switches according to network variations (Fig. 1(b)). In our implementation, we use a hardware decoder and encoder to transcode input video data to a single-rate output stream based on the user's network connectivity. Unlike adaptive streaming solutions, TransPi only provides a single transcoded stream and thus it does not provide a list of various bitrates for adaptation.

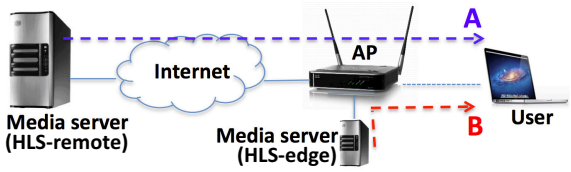


Fig. 2. (A) The video streaming is provided from a remote media server. (B) The video segments are directly downloaded from a media server at the wireless edge.

III. Inefficiency of HLS/DASH and Motivation for Transcoding

In this section, we first evaluate the performance of HLS and DASH's adaptive algorithms in terms of bitrate, bandwidth utilization, stall, and rebuffering time. We present results that show the benefits of deploying video streaming service at the wireless edge. We then motivate the necessity of video transcoding solution at the wireless edge in order to provide better streaming service to the user by addressing several key questions.

A. Bitrate Selection and Bandwidth Utilization

In adaptive bitrate streaming (HLS and DASH), the server maintains multiple profiles of the same video encoded in various bitrates and quality levels. Further, the video object is partitioned into multiple segments each of which has a duration of one second. A client player then requests different segments at different encoding bitrates, depending on the underlying network conditions and adaptation algorithm.

We created a network testbed to evaluate the performance of HLS and DASH. For our experiments, we first set up media servers that provide adaptive streaming service with HLS standards. The media server hosts pre-processed HLS video segments of the same video encoded in bitrates of 0.5, 0.8, 1, 1.5 and 2 Mbps. We deployed one such media server in a remote location where the end-to-end ping latency between a client and the server (HLS-remote) is around 58 msec (see line A in Fig. 2).

The cloudlet-based system is introduced to support services that require low-latency, such as real-time applications. By providing the service from the vicinity of the end user or the wireless edge, cloudlet architecture has multifold benefits: (i) reduce the network latency, (ii) provide more agile service adapted to the network changes and (iii) have a vantage point from which to monitor the status of the client. Cloudlet architecture would be beneficial for delay-sensitive applications, especially over wireless networks, since it could address low latency requirement and quickly incorporate network dynamics. To realize the cloudlet-based system, we have deployed the same HLS server in the wireless edge which is directly connected to the AP as depicted in Fig. 2 (red dashed line B). In this setup, the ping latency to the HLS-edge server is around 4.2 msec.

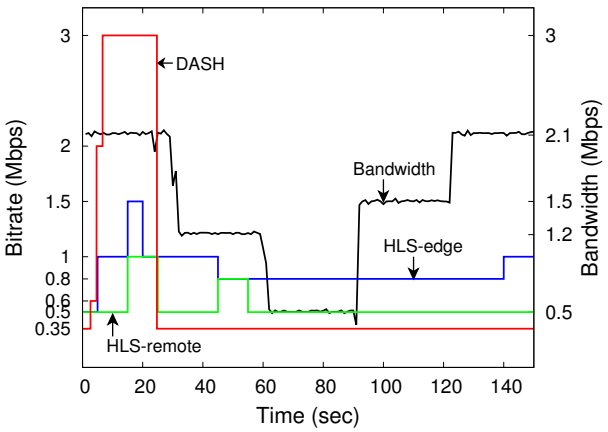


Fig. 3. The adaptive steaming solutions under-utilize the available bandwidth by selecting lower video bitrates.

The client player sends a request to the media server and adaptively downloads the video segments according to its wireless capacity and buffer level. In this set of experiments, we assume the last wireless hop is the bottleneck wherein many devices connect to the same AP in the home wireless environment and compete for bandwidth. We have used a programmable AP, Paradrop AP [22] and injected network variations by controlling the downlink capacity between the AP and the client. While the client receives video streaming service, we adjust wireless capacity with 30 second time intervals (at 2.1, 1.2, 0.5, 1.5 and 2.1 Mbps) and trigger the client player to adapt the video bitrates. The client receives video segments via the AP according to the HLS's adaptation algorithm considering the client's channel condition and profile (e.g., buffer level, network capacity and etc.).

Similarly, we have also tested the same experiment with the DASH protocol for comparison. We used the most recent client player (DASH.js [35]) and publicly available DASH test streams [34]. In DASH configuration, the available video bitrates are 0.35, 0.6, 1, 2 and 3 Mbps[1] and the ping latency from the client to the DASH stream server is ∼62 msec which is similar to that of HLS-remote.

Bitrate selection. In Fig. 3, we present the selected video bitrates of two adaptive bitrate streaming solutions, HLS and DASH, along with available network capacity bandwidth between AP and the client (black solid line). We notice that the HLS (HLS-remote and HLS-edge) conservatively under-utilizes available bandwidth, and hence the user experiences a lower video quality. That quality could have been improved, given that network bandwidth is much higher than the selected bitrates. This observation clearly corroborates findings from prior studies that compare the performance of several adaptive HTTP streaming players [15], [3]. Perhaps one might argue

[1]The tested DASH stream only provides five different bitrates. The media provider generally prepares a small number of bitrates for adaptive bitrate streaming for several reasons (storage, cost, efficiency, policy and etc.)

	Bandwidth utilization
HLS-remote	39.7%
DASH	30.6%
HLS-edge	56%

TABLE I
BOTH HLS AND DASH SIGNIFICANTLY UNDER-UTILIZE AVAILABLE BANDWIDTH, HOWEVER EDGE-BASED STREAMING COULD IMPROVE THE BANDWIDTH UTILIZATION.

that the client selects lower bitrates due to the unavailability of other bitrates, however, this might not be true. For instance, the client could have picked a video segment of 1.5 or 2 Mbps bitrate during 120-150 seconds while the available bandwidth is around 2.1 Mbps, however, 0.5 and 1 Mbps was the highest bitrate that was actually selected by HLS-remote and the HLS-edge, respectively.

In contrast, DASH aggressively selects a higher bitrate (3 Mbps) at the beginning, even though the channel bandwidth (2.1 Mbps) is not capable of downloading such segments. This causes a freeze of significant duration during playback (we will elaborate in the next subsection). After aggressive selection, DASH switched to the lowest bitrates (0.35 Mbps) regardless of channel changes. Both HLS and DASH were not able to incorporate accurate bandwidth estimation, therefore, their adaptation decisions are either too conservative (HLS) or too aggressive (DASH).

One interesting observation is that selected bitrates are higher when the client receives video segments from the media server connected to AP (HLS-edge) than they are from the remote server (HLS-remote). Note that HLS-edge, however, still under-utilizes the available bandwidth. Higher bitrates could be beneficial for users since they could provide better streaming service. There are many network factors that have an impact on the quality of video streaming service. For example, high bandwidth and low network latency are required to provide satisfactory service (e.g., no stalls, no glitches) to users. In this sense, there are two advantages of deploying video streaming service near the AP; (i) it can reduce the latency by providing service close to the user and (ii) it is agile to network changes since it can instantaneously incorporate client feedback (wireless channel condition). The performance of video streaming over wireless could be improved when it is serviced from the vicinity of the AP for the above reasons.

Bandwidth utilization. The bandwidth utilization can be inferred by selected video bitrates and the time duration of video segments. In other words, the area of bitrate plotted in Fig. 3 corresponds to the bandwidth utilization for each adaptive scheme. We obtained the average bandwidth utilization of HLS-remote, HLS-edge, and DASH by repeating ten runs with the same condition and summarized these results in Table I. It can be seen that both HLS and DASH only use less than 60% of the total network capacity. Again, we confirm that the cloudlet improves HLS's performance by 41% in terms of bandwidth utilization. We can see that the selection of lower bitrate in turn leads to the under-utilization of given network

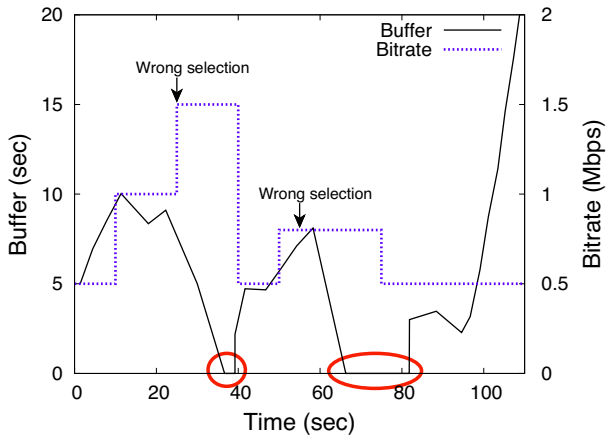


Fig. 4. Inaccurate bitrate selection leads to frequent stalls and rebuffering. The client player freezes at 36.5-39.2 and 66.3-81.8 seconds due to the buffer depletion (marked in the red circles).

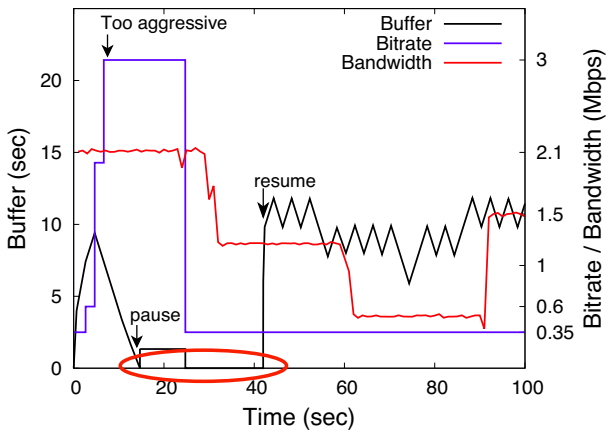


Fig. 5. DASH aggressively selects higher bitrates and eventually this causes significant stalls, 14.6-42 seconds.

capacity.

Thus, *carefully selected video bitrates are indeed important to maximize the bandwidth utilization and to provide higher streaming quality to the users. In addition, providing video streaming at the wireless edge (AP) could also help to improve the quality of video streaming.*

B. Rebuffering and Stall

Here, we focus on how bitrate selection affects video quality with respect to video freezes and rebuffering time. The client player utilizes the buffer for storing video frames ahead of time in order to prevent stalls due to unexpected networking malfunctions. Besides the network capacity, the player's buffer occupancy is one of the factors for adaptively determining the bitrates of video segments when downloading. For example, when the player's buffer level is above threshold, then the client picks higher bitrates to enhance the video quality, whereas when the buffer is low, the client selects lower bitrates to fill the buffer again. However, if the bitrate decision is not carefully managed based on buffer occupancy, then it could lead to unpleasant results such as frequent stalls and long rebuffering times.

Rebuffering. In this set of experiments, we used the same HLS remote server and controlled the wireless capacity at the AP in the same manner as described in the subsection III-A to trigger rate adaptations in the client. During the experiments, we measured the instantaneous buffer level at the client player and its bitrate decisions. Fig. 4 presents one particular example of both buffer level (in terms of seconds) and selected bitrate while streaming video on the client (we have observed the same behavior from other experiments and presented one of them for the sake of brevity). We have observed two buffer depletions from 36.5 to 39.2 and from 66.3 to 81.8 seconds as marked in circles. During these periods, the client player stalled and rebuffered for about 2.7 and 14.5 seconds, respectively. The main reason that the player stalled comes from the fact that the client selected inaccurate (too high) bitrates. Specifically, we can see that the buffer level decreases from 25 to 36.5 seconds, whereas the selected bitrate jumps from 1 to 1.5 Mbps. The aggressively selected bitrate depleted the buffer and eventually led to stall. In consequence, the client lowered the bitrate from 1.5 to 0.5 Mbps at around 40 seconds, and thus filled up the buffer again. This late switching is due to the inappropriate decision of the adaptive algorithm; an algorithm that is based on a mixture of buffer occupancy and inaccurate channel estimation. Note that the client could have avoided such stall and rebuffering by selecting the lower bitrate (e.g., 1, 0.8 or 0.5 Mbps) earlier instead of selecting 1.5 Mbps. We observed the same behavior during the second stall at around 60-81 seconds. The sudden bandwidth changes in the middle of downloading video segment could trigger the client to switch to a different version of segment. This also leads to buffer depletion because the partially downloaded segment cannot be used.

Similarly, we have run the same experiment with the DASH client and server. We present the buffer occupancy and chosen bitrates in Fig. 5. Notice that DASH's adaptation algorithm is more aggressive than that of HLS; it immediately tried the highest bitrate (3 Mbps) at the beginning (at 8 seconds) even if the bandwidth (2.1 Mbps) was lower than that bitrate. This decision made the player freeze for almost 30 seconds (marked in the red circle) until 42 seconds when the buffer replenished enough to play again.

Stall. We also present the average number of stalls and their duration as obtained from 10 runs under the same configuration in Table II. Even with different settings, e.g., different downlink capacity and bitrates of segments, we were able to find the same conclusion. The frequent stalls and the rebuffering time immediately impact the quality of streaming service and user engagements [19]. This result shows that the inaccurate bitrate adaptation used in HLS and DASH does not work well in a highly variable network, and could therefore

	Number of stalls	Rebuffering duration	Buffering ratio
HLS-remote	1.3	13.7 sec	9.1%
DASH	1	25.7 sec	17.1%
HLS-edge	0	0 sec	0%

TABLE II
AVERAGE NUMBER OF STALLS, REBUFFERING DURATION AND BUFFERING RATIO DURING MULTIPLE RUNS OF 150 SECONDS PLAYBACK.

hinder users from achieving the QoS requirement of streaming video in their wireless networks. The HLS-edge does not introduce stall or rebuffering. However, it is not able to utilize bandwidth efficiently while selecting for lower bitrates. Again, better bitrate adaptation logic and carefully selected bitrates based on accurate channel estimation could prevent video stalls and hence enhance QoE.

This in turn *motivates us to design a channel-aware bitrate decision approach that does not incur stalls or freezes. We also consider the edge-based system for streaming service given the benefits of the cloudlet in wireless networks.*

C. Motivating Video Transcoding at the Wireless Edge

The inefficiency of current bitrate adaptive solutions as discussed and shown in the previous two subsections (III-A and III-B) motivates us to design a new approach for streaming video. Several inferences from our experiment results are summarized as follows: (i) bitrate adaptive algorithms implemented on HLS and DASH are not as efficient at adapting to the network changes, given that selected bitrates are not optimal, (ii) the number of video bitrates of pre-processed segments are not sufficient to handle the network dynamics, and (iii) bitrate adaptation does not work well in the wireless environment, especially where the last hop to a client is wireless. Regarding the first inference, one of the reasons that bitrate decisions are not optimal is due to the inaccurate channel estimation and subsequent reactions based on that estimation. There are several approaches to improve the adaptation algorithm, however, they have focused on application layer solutions (e.g., accurate channel estimation and better adaptation algorithm) which do not involve changes to the media source itself. Second, it is challenging to pre-process and store numerous bitrates of segments for providing fine-grained bitrate adaptation because of the fact that there are many video codecs and platforms of diverse users, and pre-processing requires additional storage for outputs. A limited choice of bitrates frequently causes under-utilization of network capacity, and this leads to lower quality of streaming service. It is hard to pre-process live media sources (e.g., sports games, news, and teleconferences) in real-time, thus pre-processing is limited to certain purposes such as video on demand. Third, the cloudlet or edge-based system would be beneficial for video streaming service especially over wireless networks since it could address low latency requirement and quickly incorporate high variations of wireless network.

These inferences motivate us to design a channel-aware transcoding solution that directly modifies the media source on-the-fly instead of providing multi bitrates. We propose TransPi, a video transcoding system running at the wireless edge that enhances user experience with the video streaming while providing seamless streaming service to the user in a varying wireless environment accounting for the user's profile. Considering the difficulty of storing multiple versions of the same video with various bitrates in a media server, we leverage a video transcoding scheme that generates a single stream whose bitrate is instantaneously determined. Instead of adapting within given bitrates of segments, TransPi transcodes the original source to the most appropriate bitrates and provides streaming service. It thereby eliminates inefficient bitrate adaptations for the client. TransPi takes a channel-aware approach for determining the bitrates of a transcoded stream (details in Section IV). Unlike HLS or DASH, TransPi adaptively switches the video bitrates while receiving instantaneous client's bandwidth feedback from the AP, therefore it is more agile to network changes. Knowing the benefits of cloudlet architecture for video streaming service, we deploy our transcoding solution at the wireless edge where it can monitor the client's profile from the nearest vantage point in order to maximize the QoS. In addition, having the video transcoding solution at the wireless edge provides more affordability to the end user and allows speedy deployment in existing systems. Since TransPi changes the media source directly, it does not require additional storage for hosting various bitrates of segments.

IV. SYSTEM DESIGN AND IMPLEMENTATION

A. System Overview

Our system consists of four components; media source (DATN), video transcoder on Raspberry Pi (RPi) [21], bandwidth monitor running on AP and client's video player as depicted in Fig. 6. The basic operations of system are as follows:

1. On the client's video player, the client selects a channel to watch. This will trigger the transcoding process by sending a channel request to the video transcoder on RPi.
2. The bandwidth monitor running on AP periodically (100 msec) sends the client's information (e.g., downlink bandwidth) to the encoder on RPi. This information is used for determining transcoding parameters instantaneously.
3. The video transcoder fetches the requested TV stream from the media source (DATN) and then decodes and encodes the requested stream. While encoding, the video transcoder adapts the video bitrates (once every one second) for transcoding based on the received client's information.
4. The transcoded output segments are consistently delivered to the client over the TCP connection (between transcoder and client player). The client then plays them in real-time.

We describe the details of each procedure and their implementations in the following subsections.

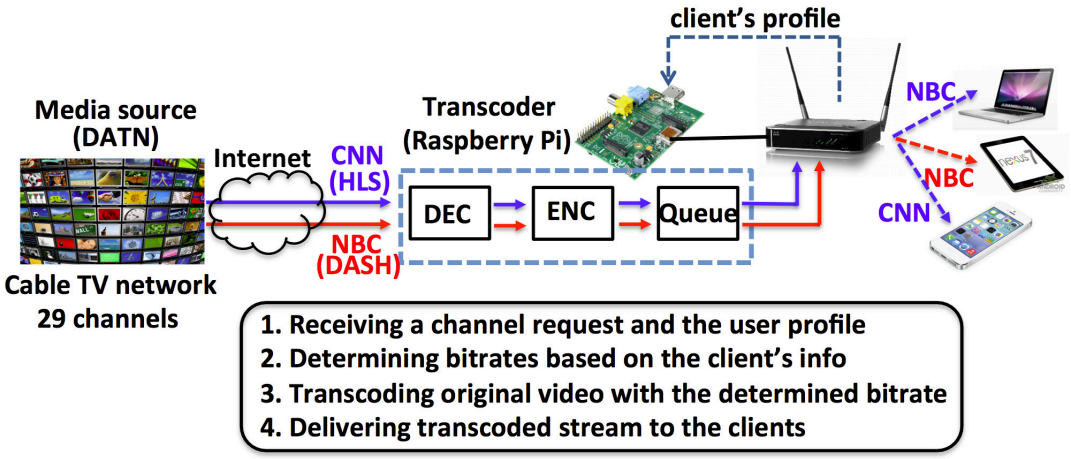


Fig. 6. TransPi interacts with the client to determine the parameters (e.g., bitrate) for video transcoding and provides seamless streaming service in real-time. TransPi switches the encoding bitrate according to changes in the network.

B. *Video Transcoding*

Four entities are involved in the video transcoding.

DATN, live media source. The University of Wisconsin-Madison campus network provides the IP-based Digital Academic Television Network (DATN) service in the area of campus dormitories and University housing so that users are able to access TV streams with their mobile devices or computers [20]. DATN is a TV network that carries 27 channels (e.g., CNN, NBC and ABC running 24/7) operated by the University. However, due to the scarce resources in wireless capacity and the high volume of demand, users may experience unsatisfactory video quality. To remedy this problem, we apply our transcoding solution on DATN channels to enhance the user experience. We make the incoming DATN TV streams to have either HLS or DASH format to support various methods in TransPi implementation, however other formats can be also incorporated.

Raspberry Pi. We use Raspberry Pi for transcoding DATN TV streams. The RPi is a low-cost, credit card sized computer that includes a graphics processing unit (GPU) and a hardware video decoder and encoder. The hardware de-/encoder can support HD (1080p) video de-/encoding with low power consumption (maximum 3.5W). The RPi provides OpenMAX APIs [38] to access the video de-/encoder for the video transcoding process. For instance, hardware de-/encoder can improve the performance of transcoding process more than 4 times faster with 18 times less CPU utilization comparing to the software-based video transcoding [14].[2]

The transcoding process is accelerated by the hardware video de-/encoder on RPi. The open source project gst-omx provides a wrapper of the OpenMAX API and can be used with the GStreamer framework [36] to develop a video transcoding application on RPi. The hardware video de-/encoder on RPi can transcode both 720p HD and one SD streams or three SD streams simultaneously. Therefore, a single RPi can provide independent transcoding service to multiple users (up to three) at the same time. We build a RPi cluster consisting of multiple RPis and a transcoding manager in order to support a large number of users (details of PRi center can be found in [41]). Note that, if multiple users share the same transcoded stream, a single RPi can support more than three users. In a typical home environment, one RPi is sufficient for providing transcoding service.

Video transcoder. We have implemented a video transcoder on RPi to utilize the hardware de-/encoder for video transcoding in TransPi. We use GStreamer, an open source multimedia framework, to develop our video transcoding application running on RPi. In the GStreamer application, different plugins are connected to form a pipeline to process media data. For instance, the HLS demux plugin is used to manage the connection with the HLS media server and download video segments while the TS (Transport Stream) demux plugin is used to separate video and audio data from the video stream. GStreamer plugins for the hardware decoder and encoder are then connected to transcode the video data. After the transcoding process, the video data and audio data are muxed again to generate the TS stream and then the client downloads that TS stream over the TCP connection. Based on basic functionality of GStreamer framework, we modified and optimized plungins to realize our on-the-fly transcoder.

The transcoding procedure is described as follows. The transcoder periodically receives the client's profile (e.g., downlink bandwidth) from the AP for determining the parameters of video transcoding. The video transcoder running on RPi first receives a TV channel request from the client and fetches the TV stream from the DATN. At this stage, the incoming TV stream is segmented (duration of one second). The transcoder then decodes the segment and encodes it with the selected

[2]Note that different hardware implementations lead to various performance gain. We observed from our experiments that the performance gain would be much higher with fine-tuning and optimization.

Algorithm 1 Bitrate decision for transcoding

1: Input: client's profile
2: Output: transcoding bitrate for next video segment
3: b_i: bitrate for client i, p_i: client i's bandwidth
4: **for** $i \in [1 : |I|]$ **do**
5: **if** $(p_i > b_i \times (1 + \alpha))$ || $(p_i < b_i \times (1 - \alpha))$ **then**
6: $b_i = p_i$, $s_i = p_i$
7: **end if**
8: **end for**
9: s_i: selected bitrate for client i

transcoding parameters (e.g., bitrate, profile, level, IDR interval). While encoding, the video transcoder instantaneously adapts the bitrates (once every second) based on the received client's information. The output of the transcoding process is pipelined to the output queue in the transcoder, then the transcoder converts it to a continuous video stream before it is delivered to the client (we maintain the output queue size of 400 msec in order to minimize the delay to the client). The client receives the transcoded stream over the TCP connection from the transcoder and plays it in real-time.

The embedded video decoder and encoder in RPi are highly efficient for real-time video transcoding. The reason TransPi is able to transcode live streaming in real-time is that it first transcodes the incoming video segments immediately and then concatenates transcoded outputs into a single stream. The delay caused by the transcoding process is very minimal; there is only about 400 msec delay for live TV streaming. TransPi shows the feasibility of low latency, on-the-fly and real-time transcoding service with live TV channels.

One problem of RPi's video encoder is that it is designed for real-time communication applications, therefore it is inadequate to support generating B-frames in H.264 format. As a result, the compression ratio is limited, and hence it leads to the degradation of video quality. We believe such issue does not affect the performance of TransPi.

Channel-aware bitrate decision. Inaccurate channel estimation due to the significant channel variation in wireless network lead to poor performance in adaptive streaming service [23]. In TransPi, video transcoding parameters have a great impact on video quality. For instance, bitrate selection is critical for efficiently utilizing available bandwidth and enhancing video quality. The AP passively monitors the wireless bandwidth of each client to determine video encoding parameters. We use the commercial off-the-shelf AP and run a python script on it to realize the bandwidth monitoring. In TransPi, we use the client's downlink bandwidth to decide the bitrates of transcoding outputs. The downlink bandwidth for each client is recorded every 100 msec and reported to the encoder on RPi as depicted in Fig. 6. Based on the received client's profile, the video encoder runs the bitrate decision algorithm as described in Algorithm 1 once every second. We set $\alpha = 0.1$ to switch the bitrate of encoding when the network bandwidth changes more than 10% compared

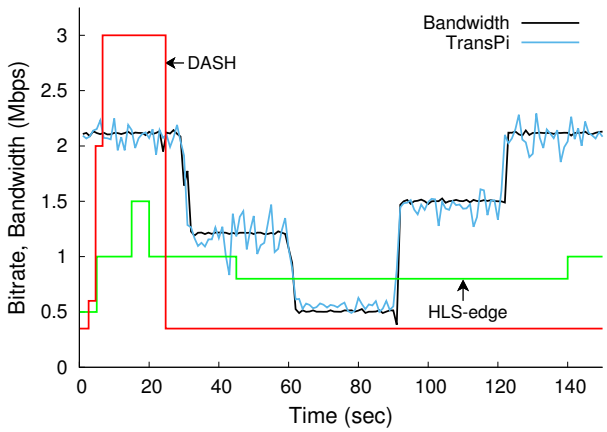


Fig. 7. TransPi's channel-aware approach keeps the transcoding bitrates close to the client's downlink capacity, thus fully utilizes bandwidth. In contrast, both the DASH and HLS-edge under-utilize the network capacity and cannot adapt to the network changes.

to the previous configuration. We have tried other parameters and bitrate selection algorithms, however their gains are not significant considering complexity. Our goal is to enhance the user experience (e.g., achieve higher bitrate and eliminate stalls and rebuffering) by providing the most appropriate video bitrates according to downlink capacity, thus Algorithm 1 is simple but effective for real-time video streaming.

C. Client Player

We tried multiple open-source video players (e.g., ffmpeg and gstreamer) on clients for playback, which does not require any modifications to play received transcoded stream. The video player on the end user's device initially creates a TCP connection to the video transcoder on RPi and requests a TV stream. The transcoded video segments are consistently delivered to the client as a single, continuous stream and the client plays them in real-time. Note that we only adapt the HLS and DASH formats for playing video on the client player without having their bitrate adaptation algorithm since TransPi only provides a single stream at a time.

V. Performance Evaluation

We have shown the inefficiency of HLS/DASH streaming solutions in section III. In this section, we present and compare the performance of TransPi in the same network setting that we evaluated HLS and DASH.

A. Micro-Benchmark

Bitrate and bandwidth utilization. To show how accurately TransPi adapts bitrates for transcoding according to the network changes, we present video bitrates of transcoded output and network bandwidth in Fig. 7. We can clearly see that the video bitrate is very close to the downlink capacity while TransPi incorporates the client's instantaneous channel condition to determine the video bitrates of transcoding. We

Time (sec)	Bandwidth	Avg. bitrate	Stdev.
0-30	2.1 Mbps	2.08 Mbps	0.08 Mbps
30-60	1.2 Mbps	1.19 Mbps	0.14 Mbps
60-90	0.5 Mbps	0.57 Mbps	0.07 Mbps
90-120	1.5 Mbps	1.43 Mbps	0.13 Mbps
120-150	2.1 Mbps	2.07 Mbps	0.13 Mbps

TABLE III
THE AVERAGE VIDEO BITRATE OF TRANSCODED OUTPUT IS VERY CLOSE TO THE NETWORK CAPACITY AND STANDARD DEVIATION IS RELATIVELY SMALL.

point out that the small variation of video bitrates of transcoded output is due to the behavior of the hardware video encoder in RPi. This video encoder sets the range of video bitrates during the encoding process instead of a constant bitrate value. For this reason, even if the measured bandwidth is stable (constant) and we set the video bitrates to constant, the video encoder is not able to set the output video bitrates to constant. The variation of bitrate across I/P-frames in a group of picture (GoP) inherently propagates to the bitrate of transcoding. As a result, this introduces some inevitable variations in output bitrates as shown in Fig. 7. We present the average bitrate and standard deviation of the transcoded stream in Table III. The averages of bitrates during each 30 second period (with respect to the network changes) are very close to that of the bandwidth and the standard deviations remain relatively small[3]. TransPi spontaneously switches the bitrates, however, this does not break (or interrupt) video playback nor affect the user experience at all because the client player seamlessly plays continuous video stream. There is no sudden scene change or freeze during playback.

We also evaluate the bandwidth utilization of TransPi and compare it to that of HLS and DASH. TransPi almost fully utilizes the available network capacity while the bandwidth utilization is 99%. We can tell from Fig. 7 that TransPi's output video bitrates are very close to downlink capacity. Compared to HLS and DASH, TransPi improves the bandwidth utilization by 150% and 224%, respectively, while TransPi has much higher bitrates than HLS or DASH. These higher bitrates of output stream can lead to higher video quality assuming the same compression ratio of the video encoder.

Buffer. The size of the transcoder output buffer eventually introduces some delay to real-time streaming while playing on a video on the client side. We maintain the output queue of the transcoder at 400 msec, which is not only enough for compensating for sudden changes in the wireless network but also sufficient for providing real-time streaming service to the clients with minimal delay (eventually TransPi adds a 400 msec delay. We empirically set the output queue to 400 msec; a lower threshold could cause stall and a higher value could introduce a longer delay, 400 msec strikes a good balance). In Fig. 8, we plot both the transcoder's output queue and the client's input buffer during 120 second playback (while

[3]In this experiment, we switch the bandwidth for every 30 seconds, however finer grained switch (e.g., 1 second) is also possible since our transcoding solution adapts video bitrates every one second

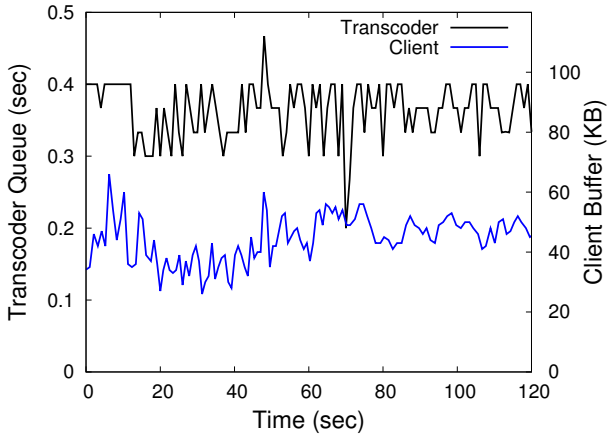


Fig. 8. The transcoder keeps the size of output queue to 400 msec to minimize the delay. The client's input buffer remains stable (30-50 Kbytes).

controlling bandwidth in the same manner). This shows that TransPi constantly keeps the output buffer level close to 400 msec while the client's buffer occupancy is slightly affected by the network changes. There are small variations (50-100 msec) in the transcoder's output buffer size; this is mainly due to the variation of bitrates across the GoP. We present the transcoder's queue size in terms of second (in Fig. 8) to highlight the minimal delay caused by the transcoder (i.e., 400 msec), however it can be represented in KB, buffer size. For instance, 300 and 400 msec correspond to 57 KB and 75 KB, respectively. The transcoder's queue size is maintained on average 67 KB during 120 second playback which is slightly higher than the client's buffer size.

The buffer size of the clients's player also remains stable (at between 30-50 KB) because the incoming data rate of video segments is equals to the playback bitrate. The player also can slightly adjust the playing speed according to the buffer size without causing any noticeable difference to the human eye; e.g., increase (decrease) the speed when the buffer level is higher (lower) than the threshold. We can observe that bandwidth changes do not affect the client's buffer level. The client's buffer neither dries out nor overflows because of that TransPi switches video bitrates according to the downlink speed. Due to this, we do not observe either stall or rebuffering time during playback.

The buffer size of the transcoder is related to the periodicity of the bitrate decisions. TransPi makes a bitrate decision once every second incorporating the client's channel information. One of the reasons for keeping a large buffer for current state-of-the-art video players is to prevent stall and rebuffering time due to network changes, however, this is not necessary in TransPi because our channel-aware bitrate decision accounts for unexpected network variations immediately. Moreover, a larger buffer would increase the delay in real-time streaming service, thus we keep the transcoder output queue low in TransPi. We plan to investigate optimizing buffer management

and the periodicity of bitrate decision for better operation in our future work.

Fairness. Significant unfairness between clients has been observed for bitrate adaptive streaming solutions when multiple clients share the network bandwidth [16]. Providing bandwidth fairness is important to guarantee the QoS in multi-client deployment. TransPi determines the bitrates for transcoding based on the client's downlink capacity, thus selected bitrates allocate the bandwidth for clients in the network. This eventually ensures fair sharing of bandwidth across multiple clients.

To evaluate the performance of TransPi under a competing scenario, we repeat similar experiments with four clients. In this setup, each client receives transcoded video streaming from a dedicated transcoder (RPi) and shares downlink bandwidth while associating with the same AP. Each transcoder incorporates its client's network bandwidth to decide the output bitrates. To evaluate the fairness under multiple clients deployment, we first set the downlink bandwidth in a symmetric manner. Fig. 9 shows the selected bitrates of four clients while applying network bandwidth changes configured to 20, 15, 10, 15 and 20 Mbps for 10 seconds each (the shared bandwidth for each client is 5, 3.75, 2.5, 3.75 and 4 Mbps during each time interval). We can clearly see that the selected bitrates for each client remain same; four clients equally share the available bandwidth, $1/n$ where n is the number of clients in the network. This ratio of bitrates for clients remains the same even when the network's bandwidth changes. The sum of selected bitrates for both clients equals to the network capacity, thus TransPi efficiently utilizes bandwidth (utilization is around 99%). In this experiment, we set the time interval to 10 seconds to characterize the network dynamics, however even smaller time interval (e.g., 1 second) is feasible, because TransPi configures the transcoding bitrate every 1 second. We omit the result for the sake of brevity.

Similarly, we control the downlink bandwidth in an asymmetric manner; client 1 receives two times the downlink bandwidth compared to other clients. We confirm that the ratio of the selected bitrates for each client stays at 2:1:1:1 as we configure the downlink bandwidth for four clients, and the ratio remains same while the network's bandwidth changes (we omit the results for the sake of brevity). In this set of experiments we have confirmed that both fair-sharing and controlled configuration are feasible in TransPi by adapting the bitrates for the clients in the network.

Providing higher video quality. Higher bitrate and efficient network utilization could yield better video quality. In addition, the Peak Signal-to-Noise Ratio (PSNR) is a standard metric of video quality and is a function of the mean square error between the original and received video frames. It is one widely used metric for evaluating video quality. However, PSNR calculation is not adequate to evaluate the adaptive bitrate scenario because of the fact that PSNR is used to measure the quality of reconstruction of lossy compression codecs and capture the pixel losses. Due to this reason, the PSNR value is not representative in evaluating the quality of

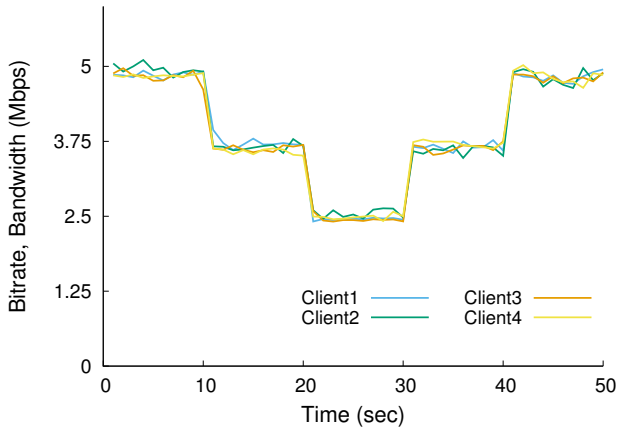


Fig. 9. Regardless of the bandwidth changes in the network, the transcoding bitrates for each client remain same. Four clients equally share the bandwidth.

adaptive streaming where the pixel difference between two video streams with different resolution (or bitrates) is almost negligible. PSNR difference is 1-2 dB when comparing the streams of 0.5 Mbps to 2 Mbps bitrate. However, we have experienced much higher video quality when watching higher bitrate streaming of TransPi compared to lower bitrates of HLS or DASH. For instance, 2.1 Mbps streaming of TransPi provides better quality of service than 0.5 or 1 Mbps of HLS or DASH at the first 30 seconds. The higher average bitrate implies higher video quality and results in increased user engagements [18]. In addition, no stall or rebuffering time in TransPi also contributes to a better experience in real-time streaming service.

B. Video Quality and User Experience

The quality of the original video is obviously higher than the transcoded one, since transcoding downgrades the video quality in terms of accuracy and resolution. There are many evaluation metrics for gauging user experience of streaming video, and video quality is one of them. However, higher video quality does not always guarantee better user experience. For example, even if the provided video quality is high, frequent stalls and long buffering time could lead to lower user engagement and satisfaction because several metrics are interrelated. This eventually lowers the quality of service to the users.

The authors in [19] observed that there is a non-monotonic relationship between video quality and user engagement (e.g., higher average video bitrate does not always results in higher user engagement). As shown in many previous researches [19], [24], [25], user engagement and quality of experience depend on many factors; video bitrate, rebuffing time, delay at startup and bitrate switches. In addition, video streaming service providers prefer to lower the quality of the video delivered rather than cause an interruption in its playback. The main purpose of utilizing video transcoding in this work is to provide better streaming service to the user when the

wireless network cannot provide successful transmission of original video data on time. TransPi can enhance the user experience by eliminating stalls, buffering and start time, therefore compensating for some loss of video quality. In addition, TransPi efficiently utilizes the available bandwidth by instantaneously adapting bitrates of transcoded stream to the wireless capacity, thus it can provide higher bitrates compared to other bitrate adaptive streaming solutions.

C. Characteristics of TransPi

The transcoding solution implemented in TransPi is unique in several ways. First, and most importantly, is that TransPi leverages a hardware assisted transcoding solution and is compatible with both the HLS and DASH standards. It provides seamless bitrate switching while transcoding video in fine time granularity, which means that end users are not interrupted during playback. It also utilizes available wireless bandwidth more efficiently because the transcoded video bitrates are determined by instantaneous bandwidth measurements. Moreover, the client does not need to wait until the whole video has been transcoded since the transcoded output is immediately delivered to the client to support real-time streaming. Unlike other transcoding systems (e.g., [4]), TransPi does not require pre-processing or additional storage for the transcoded video since TransPi provides an on-the-fly transcoding service. TransPi takes the user's profile instantaneously to determine the bitrates for transcoding, thus it quickly adapts to the user's current network condition as opposed to other bitrate adaptation solutions [2], [5] that only serve a limited choice of pre-processed bitrates. In contrast to progressive downloads, our streaming scheme barely fills up the buffer at the client, and it thereby avoids the waste of network resources caused by unviewed downloads.

Our transcoding solution can be applicable to other video streaming applications and to online video streaming with various formats. Since our transcoding system can transcode live streaming video on-the-fly with very minimal delay, the benefits can be maximized when TransPi is used with live video, such as a broadcast sports game, breaking news, and camera streaming. Recall that TransPi's bitrate switches do not break video playback and it provides seamless adaptation.

VI. Related Work

Adaptive streaming. Adaptive bitrate streaming is a technique used in streaming video over HTTP where various versions of the source video are pre-encoded at different bitrates [10]. Dynamic adaptive streaming over HTTP (DASH) helps to maintain user perceived quality while adapting the video quality based on the available network bandwidth. Several measurement studies [3], [2] presented the in-efficient performance problems of major players, e.g., Smooth streaming, Netflix, and Adobe HTTP Dynamic Streaming. They show that none of these players is good enough; they are either too conservative or too aggressive and hence they do not efficiently adapt to the network variations. The authors in [23] show the difficulty of accurately estimating capacity and propose a rate adaptation algorithm based on the instantaneous playback buffer at the client.

Several papers [15], [16] analyze the clients behavior with respect to efficiency, fairness and stability when multiple clients are in the same network contention domain. Inaccurate bandwidth measurements caused by the temporal overlap of downloading chunk causes under/over-estimation of underlying bandwidth and results in incorrect bitrate selections. The authors in [16] present a guideline for designing better scheduling and bitrate selection logics in the client player, however, such design requires significant modifications to the client and hence hinders broad deployments. Similarly, we have observed in-efficiency of the rate adaptation mechanism implemented on HLS and DASH. As described, there are many performance issues on widely used adaptive streaming solutions, thus work in [37] proposed two rate adaptation algorithms of serial and parallel fetching methods for DASH. The main difference from such works is our system involves changing the media source itself through video transcoding instead of having better adaptations. Odyssey [42] proposed a general application-aware adaptation framework for mobile devices. The proposed solution is very similar to our system. Rather than running the adaptation on a mobile device, we propose to deploy the transcoding solution at the wireless edge (e.g., WiFi AP) that is close (just one hop) to the mobile devices.

Video transcoding. Much of recent research on video transcoding has been in the context of cloud platforms [8], [4]. The authors in [8] proposed a cloud-based proxy for delivering high quality transcoded videos while allowing adaptation to network dynamics. They built a framework for parallelizing multi-level transcoding and showed the efficacy of their system on a cloud testbed. Li *et al.* [4] presented cloud transcoder which takes user-specified parameters and transcodes videos to various formats (e.g., bitrate, resolution and codec) to the user. They store all transcoded videos and their metadata, and leverage cloud cache for reusing the transcoded video in case other users request the same video again. Cloud transcoder requires additional storage in the cloud for saving such transcoded videos for future usages. In contrast, TransPi does not require storage for transcoded video since it handles transcoding on-the-fly and directly delivers streaming video to the end user. In addition, we can minimize the network latency by deploying our transcoding solution near the WiFi AP.

Edge computing. Mobile cloud computing (MCC) has become a promising technology for mobile services while establishing a cloud infrastructure featured with vast computational resources and power [26], [30]. The authors in [28] proposed the concept of cloudlets to handle the high latency between the client and the resources in the cloud. Researches in [32], [33] showed the importance and benefits of offloading computation to nearby cloudlets for resource intensive applications. Fesehaye et al. [31] evaluated the performance of applications exploiting nearby resources in the cloudlet and showed that cloudlet service outperforms that of typical cloud-based approach. Especially, edge computing could reduce

transfer latency and increase throughput for interactive mobile cloud applications such as video streaming, file editing and chatting [31].

Hardware video encoding. Video encoding is a computationally intensive task for general purpose processors. Hameed *et al.* have shown that ASIC implementation of H.264 encoder could improve power efficiency up to 500× compared to the software-based implementation running on general purpose processors[40]. In spite of the power efficiency, hardware video encoder is not flexible. Video encoders on FPGA strikes a good balance between them, in terms of efficiency and flexibility.

VII. Discussion

Performance. We have implemented our hardware-assisted video transcoding solution using a commercial off-the-shelf device, Raspberry Pi (RPi). The performance of hardware transcoding clearly outperforms the software-based solution. However, a single RPi can handle a very limited number of video transcoding processes in parallel due to the limitation of its GPU and a hardware decoder and encoder. For example, the video decoder and encoder in RPi can transcode one 720p HD (high definition) and one SD (standard definition) streams or three SD streams simultaneously. Such a hardware limitation requires multiple RPis in order to support a large number of users at the same time. To overcome such scalability issue, we build an RPi cluster (a rack of RPis) and transcoding manager which manages multiple RPi for transcoding various DATN TV channels and supports numerous users simultaneously (having tens of RPi is still cheaper than one dedicated transcoding server). We parallelize the multiple transcoding processes to support various requests from multiple users. The RPi cluster can provide video transcoding service to hundreds of users concurrently.

As described in Section IV, we have optimized the GStreamer pipelines for efficient video transcoding, but there are certain limitations that our software implementation cannot Overcome. We are in the middle of the process for integrating our video transcoding solution into the AP which has a powerful graphics processor unit (GPU).

Caching. TransPi provides video transcoding service on-the-fly and does not store transcoded video for future usage. Reusing transcoded video streams could be economical, given that the transcoding process requires high computational power and resource. For example, popular video clips on YouTube that have many views could be reused after transcoding. Incorporating transcoded videos with cloudlet storage, we can integrate the caching scheme to reduce the workload on the transcoder server and increase the reusability of transcoded videos. The amount of data traffic traversing the core network can be reduced by deploying caches at the network edge. Moreover, latency will also be reduced with the optimized caching mechanism.

Limitations. We have shown the design of TransPi and present its performance improvement comparing with other bitrate adaptive solutions, however, there are still some limitations. i) TransPi provides the best performance when the last hop wireless link is the bottleneck in the network because of the fact that it instantaneously incorporate client's channel feedback for quick adaptation. In practice, we can easily find such use cases, e.g., public WiFi networks, cellular networks, home networks with channel quality issues due to high variation and wireless interference caused by microwaves, Bluetooth devices, walls, etc.. However, if the link between the transcoder and source is the bottleneck, e.g., home network has a relatively slow speed for its broadband link while the WiFi AP supports faster protocol (802.11ac), then TransPi no longer provides the same performance improvement in its current form because TransPi focuses on the wireless edge. ii) We build a RPi cluster to overcome a scalability issue, however the problem still remains. Specifically, TransPi works best when there are smaller number of users compared to the number of streams. However, if the number of users exceeds the number of streams (e.g., many users watch same video), then there is a point where per-user transcoding (our solution) starts to become expensive even with a RPi cluster. In this case, transcoding to multiple rates and reusing them across users may be a cost-effective solution instead of user specific transcoding. We plan to address above-mentioned two limitations in our future work.

VIII. Conclusions

We present the design, implementation and evaluation of TransPi for enhancing the performance of video streaming service to wireless users. TransPi leverages a cost-effective video transcoding solution on a low-cost hardware, Raspberry Pi, running at the wireless edge where it provides agile bitrate switches in response to network changes. We have demonstrated its superiority over bitrate adaptive streaming approaches through experiments in various wireless environments. TransPi provides quick bitrate adaptation under unstable wireless conditions and efficiently utilizes available bandwidth. It is also capable of transcoding live streaming, on-the-fly, with minimal delay. Our solution is transparent, low-cost, and scalable, thus it allows broad and quick deployment in home WiFi AP or cellular base-station.

IX. Acknowledgement

We are grateful to our shepherd, Kaustubh Joshi, and the anonymous reviewers whose comments helped bring the paper to its final form. All authors are supported in part by the US National Science Foundation through awards CNS-1555426, CNS-1525586, CNS-1405667, CNS-1345293, and CNS-1343363. This work was partly supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIP) (No.B0717-16-0036).

References

[1] J. Erman, A. Gerber, K. Ramadrishnan, S. Sen and O. Spatscheck, "Over the Top Video: The Gorilla in Cellular Networks". In Proc. of *ACM IMC*, 2011.

[2] C. Muller, S. Lederer and C. Timmerer, "An Evaluation of Dynamic Adaptive Streaming over HTTP in Vehicular Environments". In Proc. of *ACM MoVid*, 2012.

[3] S. Akhshabi, A. C. Begen and C. Dovrolis, "An Experimental Evaluation of Rate-Adaptation Algorithms in Adaptive Streaming over HTTP". In Proc. of *ACM MMSys*, 2011.

[4] Z. Li, Y. Huang, G. Liu, F. Wang and Z. Zhang, "Cloud Transcoder: Bridging the Format and Resolution Gap between Internet Videos and Mobile Devices", In Proc. of *ACM NOSSDAV*, 2012.

[5] T. Stockhammer, "Dynamic Adaptive Streaming over HTTP: Standards and Design Principles", In Proc. of *ACM MMSys*, 2011.

[6] S. Chattopadhyay, L. Ramaswamy and S. Bhandarkar, "A Framework for Encoding and Caching of Video for Quality Adaptive Progressive Download", In Proc. of *ACM Multimedia*, 2007.

[7] H. Schwarz, D. Marpe and T. Wiegand, "Overview of the Scalable Video Coding Extension of the H.264/AVC Standard", *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 17, no. 9, pp. 1103-1120, Sep. 2007.

[8] Z. Huang, C. Mei, L. E. Li and T Woo, "CloudStream: Delivering High-quality Streaming Videos through a Cloud-based SVC Proxy", In Proc. of *IEEE Infocom*, 2011.

[9] HTTP Live Streaming (HLS), IETF Internet-Drafts 2014, http://tools.ietf.org/html/draft-pantos-http-live-streaming-13/

[10] ISO/IEC 23009-1, MPEG Dynamic Adaptive Streaming over HTTP (DASH), http://dashif.org/mpeg-dash/

[11] Microsoft Smooth Streaming, http://www.microsoft.com/silverlight/smoothstreaming/

[12] Adobe Dynamic Streaming, http://www.adobe.com/products/hds-dynamic-streaming.html/

[13] Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2012-2017, http://www.cisco.com/

[14] GPU CUDA acceleration, http://www.bdlot.com/resource/gpu-acceleration.htm/

[15] S. Akhshabi, L. Anantakrishnan, C. Dovrolis and A. C. Begen, "What Happens when HTTP Adaptive Streaming Players Compete for Bandwidth?", In Proc. of *ACM NOSSDAV*, 2012.

[16] J. Jiang, V. Sekar and H. Zhang, "Improving Fairness, Efficiency, and Stability in HTTP-based Adaptive Video Streaming with FESTIVE", In Proc. of *ACM CoNEXT*, 2012.

[17] Smartphone shipments surpass PCs, http://www.pcmag.com/article2/0,2817,2379665,00.asp/

[18] A. Balachandran, V. Sekar, A. Akella, S. Seshan, I. Stoica and H. Zhang, "A Quest for an Internet Video Quality-of-Experience Metric", In Proc. of *ACM HotNets*, 2012.

[19] F. Dobrian, V. Sekar, A. Awan, I. Stoica, D. A. Joseph, A. Ganjam, J. Zhan and H. Zhang, "Understanding the Impact of Video Quality on User Engagement", In Proc. of *ACM Sigcomm*, 2011.

[20] Digital Academic Television Network (DATN), https://it.wisc.edu/services/datn/

[21] Raspberry Pi, http://www.raspberrypi.org/

[22] Paradop AP, https://www.paradrop.io/

[23] 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 *ACM Sigcomm*, 2014.

[24] S. S. Krishnan and R. K. Sitaraman, "Video Stream Quality Impacts Viewer Behavior: Inferring Causality Using Quasi-Experimental Designs", In Proc. of *ACM IMC*, 2012.

[25] P. Ni, R. Eg, A. Eichhorn, C. Griwodz and P. Halvorsen, "Flicker Effects in Adaptive Video Streaming to Handheld Devices", In Proc. of *ACM MM*, 2011.

[26] H. Dinh, C. Lee, D. Niyato and Ping Wang, "A Survey of Mobile Cloud Computing: Architecture, Applications, and Approaches", *Wireless Communications and Mobile Computing*, vol. 13, no. 18, pp. 1587-1611, Dec. 2013.

[27] M. Satyanarayanan, Z. Chen, K. Ha, W. Hu, W. Richter and P. Pillai, "Cloudlets: at the Leading Edge of Mobile-Cloud Convergence", In Proc. of *MobiCASE*, 2014.

[28] M. Satyanarayanan, P. Bahl, R. Caceres and N. Davies, "The Case for VM-Based Cloudlets in Mobile Computing", *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14-23, Oct. 2009.

[29] S. Gebert, R. Pries, D. Schlosser and K. Heck, "Internet Access Traffic Measurement and Analysis", In Proc. of *ACM TMA*, 2012.

[30] Q. Zhang, L. Cheng and R. Boutaba, "Cloud Computing: State-of-the-art and Research Challenges", *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 7-18, May 2010.

[31] D. Fesehaye, Yunlong Gao, K. Nahrstedt and Guijun Wang, "Impact of Cloudlets on Interactive Mobile Cloud Applications", In Proc. of *IEEE International Enterprise Distributed Object Computing Conference (EDOC)*, 2012.

[32] S. Clinch, J. Harkes, A. Friday, N. Davies and M. Satyanarayanan, "How Close is Close Enough? Understanding the Role of Cloudlets in Supporting Display Appropriation by Mobile Users", In Proc. of *IEEE PerCom*, 2012.

[33] K. Ha, P. Pillai, G. Lewis, S. Simanta, S. Clinch, N. Davies and M. Satyanarayanan, "The Impact of Mobile Multimedia Applications on Data Center Consolidation", In Proc. of *IEEE International Conference on Cloud Engineering (IC2E)*, 2013.

[34] DASH test stream, http://demo.unified-streaming.com/video/ateam/ateam.ism/ateam.mpd/

[35] DASH Reference Client 1.1.2, https://github.com/Dash-Industry-Forum/dash.js/

[36] Gstreamer, open source multimedia framework, http://gstreamer.freedesktop.org/

[37] C. Liu, I. Bouazizi, M. Hannuksela and M. Gabbouj, "Rate Adaptation for Dynamic Adaptive Streaming over HTTP in Content Distribution Network", *Image Communication*, vol. 27, no. 4, pp. 288-311, Apr. 2012.

[38] OpenMAX, https://www.khronos.org/openmax/

[39] Netflix encodes each movie 120 times, https://gigaom.com/2012/12/18/netflix-encoding/

[40] R. Hameed, W. Qadeer, M. Wachs, O. Azizi, A. Solomatnikov, B. C. Lee, S. Richardson, C. Kozyrakis and M. Horowitz, "Understanding Sources of Inefficiency in General-Purpose Chips", *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3, pp. 37-47, 2010.

[41] P. Liu, J. Yoon, L. Johnson and S. Banerjee, "Greening the Video Transcoding Service with Low-Cost Hardware Transcoders", *USENIX Annual Technical Conference*, 2016.

[42] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn and K. R. Walker, "Agile Application-Aware Adaptation for Mobility", *ACM Symposium on Operating System Principles*, 1997.