The following publication J. Li, Z. Peng and B. Xiao, "Smartphone-assisted smooth live video broadcast on wearable cameras," 2016 IEEE/ACM 24th International Symposium on Quality of Service (IWQoS), Beijing, China, 2016, pp. 1-6 is available at https://doi.org/10.1109/IWQoS.2016.7590439.

Smartphone-Assisted Smooth Live Video Broadcast on Wearable Cameras

Jiwei Li, Zhe Peng, Bin Xiao
The Hong Kong Polytechnic University

***Abstract*—Wearable cameras require connecting to cellular-capable devices (e.g., smartphones) so as to provide live broadcast services for worldwide users when Wi-Fi is unavailable. However, the constantly changing cellular network conditions may substantially slow down the upload of recorded videos. In this paper, we consider the scenario where wearable cameras upload live videos to remote distribution servers under cellular networks, aiming at maximizing the quality of uploaded videos while meeting the delay requirements. To attain the goal, we propose a dynamic video coding approach that utilizes dynamic video recording resolution adjustment on wearable cameras and Lyapunov based video preprocessing on smartphones. Our proposed resolution adjustment algorithm adapts to network condition changes, and reduces the overheads of video preprocessing. Due to the property of Lyapunov optimization framework, our proposed video preprocessing algorithm delivers near-optimal video quality while meeting the upload delay requirements. Our evaluation results show that our approach achieves up to 50% reduction in power consumption on smartphones and up to 60% reduction in average delay, at the cost of slightly compromised video quality.**

I. Introduction

Wearable cameras are becoming increasingly popular among people due to their portability and video recording capability [1]. GoPro, a leading company in the area of wearable cameras, has released a series of products that are equipped with Wi-Fi modules and support HTTP Live Streaming Protocol [2]. Thus, these GoPro cameras allow people to conveniently live broadcast what they see to the world, when Wi-Fi is available. In fact, wearable cameras are often used at outdoor environments where Wi-Fi is unavailable, thus requiring cellular-capable devices (e.g., smartphones) to gain network access. For example, the Livestream app for iOS has just announced its latest update that allows people to broadcast live using GoPro Hero® as a camera source for outdoor events [3].

Live broadcast audience usually expects little to no buffering while watching, and expects the video quality to be as high as possible [4]. However, our motivation experiment results show that cellular network bandwidth is constantly changing, and therefore existing approaches that set video recording resolution to a fixed value either deliver undesirably low video quality, or cause substantially long buffering. On the other hand, according to the commonly adopted HTTP live streaming protocol, recorded videos are divided into small segments in order to facilitate their delivery via HTTP [2]. This inspires a solution that video recording resolution for each video segment can be set separately based on available network bandwidth. To our knowledge, how to set the video recording resolution and whether it improves user experience remain unexplored.

In this paper, we consider the scenario where wearable cameras upload live videos to remote distribution servers via smartphones under cellular networks. We aim at maximizing the quality of uploaded videos while meeting the delay requirements of their arrivals on servers, which we mathematically formulate as an online non-linear integer optimization problem. Yet, there exist mainly two challenges. **First**, as smartphones are used to assist wearable cameras to upload videos, the incurred power consumption may greatly shorten the battery life of smartphones, which is highly undesirable in reality. How to minimize the incurred power consumption on smartphones remains a difficult challenge. **Second**, unexpected network condition deterioration [5] may cause severe time drift issue where previously recorded video segments fail to be uploaded and the transmission queue starts to grow, as new video segments are recorded. As future network conditions are unknown, adjusting video recording resolution alone may not be able to mitigate the time drift issue.

To this end, we propose a dynamic video coding approach that utilizes dynamic video recording *resolution adjustment* on wearable cameras and Lyapunov based *video preprocessing* on smartphones. **First**, we devise a resolution-maximizing algorithm to guide the resolution adjustment decisions on wearable cameras. It monitors the video transmission queue and historical upload speeds. In order to maximize the video quality, the algorithm does not adjust the video recording resolution, until historical upload speeds are consistently greater (or smaller) than the bit rate of current recorded videos. **Second**, we utilize the computing power of smartphones to preprocess (transcode) untransmitted video segments in case of network condition deterioration [6]. Based on Lyapunov optimization framework [7], untransmitted video segments are preprocessed so that they can be uploaded before their specified deadline under current network bandwidth. Due to the property of the framework, this preprocessing based approach can achieve an optimal tradeoff between video quality and upload delays.

These two techniques applied on wearable cameras and smartphones well complement each other towards two goals, namely maximizing video quality while meeting delay constraints, and minimizing energy consumption on smartphones. On the one hand, the resolution-maximization algorithm adjusts the video recording bit rate to an appropriate value, as realtime network bandwidth changes. This ensures the maximum possible video quality under the network conditions, and

also lowers the chances of preprocessing video segments on smartphones. Without the resolution-maximization algorithm, most video segments may have a high bit rate that cannot be accommodated by current network bandwidth, thus requiring being preprocessed. On the other hand, preprocessing video segments on smartphones can well resolve the time drift issue so as to meet the delay time requirements. Without using preprocessing, the resolution-maximization algorithm may cause substantial delays on video segments under deteriorated networks, which may lead to frequent playback pauses on audience side.

We evaluate our proposed approaches in real-world experiments. Results show that resolution adjustment can substantially lower the overheads of video preprocessing on smartphones due to reduced invocation times, compared to approaches that preprocess videos but do not adjust recording resolution. The preprocessing technique is also shown to effectively help reduce the live broadcast delay, compared to approaches that do not preprocess videos. Specifically, our approach achieves up to 50% reduction in power consumption on smartphones and up to 60% reduction in average delay, at the cost of slightly compromised video quality.

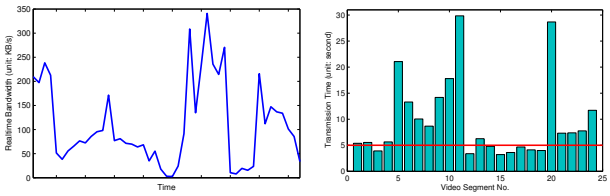
The rest of the paper is organized as follows. First, we formulate the problem in Section II. Then, we present our dynamic video recording algorithms and the preprocessing technique in Section III. Real-world evaluations can be found in Section IV, respectively. We review recent work in Section V. Finally, we conclude the paper in Section VI.

II. Motivation & Problem Formulation

A. Motivation

To motivate our work, we conducted a real-world evaluation of existing approaches for live broadcast under cellular networks. We used a GoPro camera and an Android developer phone named Nexus 6 to carry out the experiments on our university campus. The GoPro camera was set up to record videos, and was connected with Nexus 6 via Wi-Fi. Since the GoPro camera used HTTP Live Streaming Protocol, recorded videos were divided into five-second segments, and were uploaded to remote servers via Nexus 6's cellular module. In existing approaches, the video recording resolution for each video segment is set to a fixed value, regardless of network conditions. In our experiments, we set the recording resolution to 144p, 360p, 480p, 720p and 1080p, respectively, for each live broadcast session. We recorded down the cellular bandwidth and the transmission time for each video segment.

To save space, we only show the experiment results of the existing approach that sets video recording resolution to 480p in Figure 1. The recorded realtime bandwidth as shown in Figure 1a shows that cellular network bandwidth is constantly changing, which leads to distinctly different transmission times of video segments as shown in Figure 1b. The horizonal line in Figure 1b represents the maximum allowable transmission time (five seconds) under which no buffering is caused. We observe that video segments No. 5-11, 13, and 20-24 caused buffering on audience side due to their long transmission time.



(a) Recorded Realtime Bandwidth

(b) Recorded Transmission Time

Figure 1: Motivation experiment results. In the existing approach that sets video recording resolution to 480p, almost half of video segments were observed to cause buffering on audience side.

Combining other experiment results, we argue that existing approaches that set video recording resolution to a fixed value either deliver undesirably low video quality, or cause substantially long buffering.

B. System Model

We consider a scenario where wearable cameras serve as the source of video recording and recorded videos are broadcast live to clients under the assistance of connected smartphones and remote servers. Specifically, recorded videos are encoded and encapsulated for transmission on wearable cameras. Then, they are broken into a series of media files, which are transferred from wearable cameras to smartphones via their Wi-Fi connections, and then to remote servers via smartphones' cellular interface. Remote servers broadcast these media files to subscribed clients. The goal is to optimize the user experience of watching live streaming in terms of video quality and delay. Figure 2 depicts a simple illustration of our system model.

1) Video quality: How videos are recorded can significantly affect the video quality in live streaming. Since recorded videos are encoded and broken into small media files, the video recording parameters (resolution and frame rate) can be dynamically adjusted on wearable cameras for each media file, which we rename as *unit video*. We define that a unit video is the smallest unit for recording and transmission in live streaming, and has the same durance T_{unit} as all other unit videos.

We assume that there are totally n unit videos in the entire live streaming. Let $(t_1, t_2, \ldots, t_n)$ denote the beginning time of recording each unit video. Since all unit videos have equal duration, we have $t_i - t_{i-1} = T_{unit}$, for $i = 2, \ldots, n$. To simplify the model, we do not consider the frame rate when evaluating the video quality, and assume that it is constantly 30 fps. Let q_i denote the video recording resolution for each unit video. Then, we have the **average resolution** $\overline{q} = \frac{\sum_i^n q_i}{n}$ in live streaming. Setting unit videos to different resolutions results in different file sizes, denoted by $(S_1, S_2, \ldots, S_n)$. For simplicity, the same resolution for different unit videos is assumed to yield the same file size for each, and the relation between file size and resolution is simplified as quadratic. Based on file size for each unit video, we also define the **average throughput** $\overline{S} = \frac{\sum_i^n S_i}{n}$.

Figure 2: System Model

2) Unit video delay: Delay in live streaming can significantly affect the user experience. Recall that recorded videos must be transferred from wearable cameras to remote servers via smartphones' cellular interface. It is thus likely that delay occurs during live streaming due to overlarge video recording resolution or suddenly worsened network conditions. As a consequence, clients may be forced to wait for the transmission of at least one complete unit video before resuming watching live streaming.

We define the unit video delay as the time gap between the actual arrival time and the expected arrival time of a unit video. Let $(t_1^{end}, t_2^{end}, \ldots, t_n^{end})$ and $(t_1^{exp}, t_2^{exp}, \ldots, t_n^{exp})$ denote the actual and the expected arrival time of each video unit on remote servers, respectively. We say that the expected arrival time of the first unit video t_1^{exp} marks the expected starting time of live streaming, which is usually *given* in advance. Except the first unit video, the expected arrival time of a unit video is dependent on that of its previous unit video as well as its actual arrival time, i.e., $t_i^{exp} = t_{i-1}^{exp} + T_{unit} + \max(0, t_{i-1}^{end} - t_{i-1}^{exp})$, for $i = 2, \ldots, n$. The addition of $\max(0, t_{i-1}^{end} - t_{i-1}^{exp})$ takes into account, if any, the delays caused by previous unit videos. Now, we are in a position to calculate the **unit video delay** for each unit video, i.e., $T_i = \max(0, t_i^{end} - t_i^{exp})$, for $i = 1, \ldots, n$. We also define the **average unit video delay** $\overline{T} = \frac{\sum_i^n T_i}{n}$ that reflects the overall performance in terms of delays.

C. Problem Formulation

Based on the above model, we formulate the problem of maximizing the average video resolution of live streaming subject to delay constraints as follows,

$$\begin{aligned} \underset{\{q_1, q_2, \ldots, q_n\}}{\text{maximize}} \quad & \frac{\sum_i^n q_i}{n} \\ \text{subject to} \quad & \overline{T} \leq T_{m_1}, \\ & T_i \leq T_{m_2}, \text{ for } i = 1, \ldots, n, \\ & q_i \in \Phi, \text{ for } i = 1, \ldots, n, \end{aligned}$$

where T_{m_1} is the maximum tolerable average delay, and T_{m_2} is the maximum tolerable unit video delay. According to settings in common video streaming, we have $\Phi = \{144p,\ 240p,\ 360p,\ 480p,\ 720p,\ 1080p\}$. In other words, the video recording resolution must be chosen from the resolution set Φ. Note that we do not know how the cellular network bandwidth will change in the future. In other words, the entire inputs are not available from the start. Thus, the formulated problem is an online nonlinear integer optimization problem, which is very difficult to solve in practice.

III. Dynamic Video Recording & Video Preprocessing

In this section, we present the two techniques applied in our live broadcast system, i.e., dynamic video recording on wearable cameras, and Lyapunov based video preprocessing on smartphones. These two techniques combine to achieve two goals, namely maximizing video quality while meeting delay constraints, and minimizing energy consumption on smartphones.

A. Dynamic Video Recording on Wearable Cameras

One viable solution to the formulated problem is to dynamically adjust the video recording resolution on wearable cameras in order to adapt to unstable network conditions. To adapt to network condition changes, we propose a resolution-maximizing algorithm that gradually adjusts the video recording resolution. It does not change the video recording resolution until a predefined number of consecutive resolution estimates are consistently larger or smaller than the current recording resolution. When necessary, the resolution-maximizing algorithm only changes the current resolution to either its next higher level or lower level. As a result, the resolution-maximizing algorithm can fully utilize the available network bandwidth without incurring too much delay time.

Alg. 1 formally presents our resolution-maximizing algorithm when the estimate $\hat{q}_i|_{max}$ is higher than current recording resolution. The estimate $\hat{q}_i|_{max}$ is obtained based on historical upload speeds and accumulated delay time. To save space, the derivation of $\hat{q}_i|_{max}$ is not shown, and the other half of the algorithm is not presented when $\hat{q}_i|_{max}$ is lower than current recording resolution. We explain some lines of the algorithm as follows.

To satisfy the condition (line 2), the estimated maximum resolution must be larger than or equal to the weighted current resolution, and the current resolution setting is not 1080p. Note that w_H is a predefined parameter. We define a variable named `lastTimeHigh` to record the last time (line 8) when the condition (line 2) is satisfied. Only consecutive estimates that are higher than current resolution can trigger the change of the recording resolution. Thus, we define a variable named `countHigh` to track the number of such consecutive estimates. Only when `countHigh` is larger than or equal to a predefined parameter c_H, is the current resolution changed to its next higher level; otherwise, the current resolution remains (line 10-16).

Time drift issue. The inherent limitation that exists in the above proposed algorithm is the time drift issue caused by the constantly changing network bandwidth. When recorders are mobile, recorded videos may not be transmitted at the supposed time, because unexpected network bandwidth deterioration extends the necessary transmission time for previously recorded videos. At such a situation, delays may still keep increasing even if our algorithm lowers the resolution for current and subsequent unit videos. Adjusting video recording resolution can adapt to network condition changes, but may still fail to resolve the time drift issue.

Algorithm 1: Resolution-maximizing Algorithm

Input : The estimation $\hat{q}_i|_{max}$, current resolution q_{i-1}
Output: The next resolution decision q_i

$lastTimeHigh = -1, \quad countHigh = 0;$
if $(\hat{q}_i|_{max} \geq q_{i-1} * w_H)$ && $(q_{i-1} \neq 1080)$ **then**
  if $(currentTime - lastTimeHigh \leq 10)$ || $(lastTimeHigh == -1)$ **then**
    $countHigh++;$
  else
    $countHigh = 1;$
  end
  $lastTimeHigh = currentTime;$
end
if $countHigh \geq c_H$ **then**
  $q_i = q_{i-1}$'s next higher level;
  $countHigh = 0;$
  $lastTimeHigh = -1;$
else
  $q_i = q_{i-1};$
end

B. Lyapunov Based Video Preprocessing on Smartphones

The increasingly powerful capabilities of smartphones make it possible to perform compute intensive tasks for wearable cameras. For live broadcast from wearable cameras, smartphones not only serve as a bridge connecting wearable cameras and remote servers, but also can be used to preprocess videos for wearable cameras. As videos are queued on smartphones for transmission, it is possible to preprocess them so that the preprocessed videos can still be uploaded smoothly under worsened network bandwidth. A simple preprocessing technique is video transcoding, which converts the resolution of a unit video to another, resulting in a substantial change of its file size.

In order to overcome the time drift issue, we introduce the preprocessing technique to live broadcast, and propose a Lyapunov based algorithm to optimize the use of preprocessing. This algorithm helps determine whether it is necessary to preprocess unit videos in the transmission queue on smartphones, as well as how to preprocess them if necessary. It adopts the Lyapunov optimization framework that achieves some optimization objective by minimizing the Lyapunov drift-plus-penalty.

1) Derivation of Lyapunov Based Algorithm: In the following, we present the Lyapunov based video preprocessing algorithm. Let A_i denote the amount of data that arrive in the timeslot (t_i, t_{i+1}). Data are generated only during the process of video recording, which starts at time t_1 and ends at time t_{n+1}. We model A_i as the output of a function, i.e.,

$$A_i = \begin{cases} S_i & \text{if } 1 \leq i \leq n, \\ 0 & \text{otherwise,} \end{cases}$$

where S_i is the file size of the ith unit video. Let C_i denote the reduced data size in the timeslot (t_i, t_{i+1}) due to preprocessing. Let Q_i denote the queue backlog (number of bits in queue) at time t_i. Then, we model C_i as the output of a function, i.e.,

$$C_i \triangleq G(\sigma_i, \overline{B}_i, Q_i, \phi_i),$$

where σ_i represents the preprocessing method used in the timeslot (t_i, t_{i+1}), $\overline{B}_i$ is the average bandwidth achieved in the timeslot (t_i, t_{i+1}), and ϕ_i is the reduced resolution due to preprocessing in the timeslot (t_i, t_{i+1}). We define that a specific preprocessing method targets at unit videos of certain resolution, and transcodes them into a certain resolution.

Over time, the queue backlog evolves as follows:

$$Q_{i+1} = \min(Q_i - \overline{B}_i T_{unit} - C_i, 0) + A_i, \tag{1}$$

where T_{unit} is the duration of a timeslot. To remove the nonlinear operator in (1), we define a variable:

$$\beta_i = \begin{cases} \overline{B}_i T_{unit} & \text{if } \overline{B}_i T_{unit} \leq Q_i - C_i, \\ Q_i - C_i & \text{otherwise.} \end{cases}$$

Now, the equation (1) becomes

$$Q_{i+1} = Q_i - \beta_i - C_i + A_i. \tag{2}$$

The queue backlog Q_m at time t_m must be equal to zero in order to satisfy the overall delay constraint, where t_m refers to the maximum tolerable deadline for live streaming.

According to the Lyapunov optimization framework, we define the Lyapunov function as:

$$L(Q_i) \triangleq \frac{1}{2} Q_i^2,$$

and the one-step Lyapunov drift as:

$$\Delta(Q_i) \triangleq \mathbb{E}\{L(Q_{i+1}) - L(Q_i) | Q_i\}.$$

From (2), we have

$$\begin{aligned} \frac{1}{2} Q_{i+1}^2 \leq & \frac{1}{2}(Q_i^2 + \beta_i^2 + C_i^2 + A_i^2) \\ & - Q_i(\beta_i + C_i - A_i) + \beta_i C_i, \end{aligned} \tag{3}$$

since A_i, β_i and C_i are non-negative. Take expectation with respect to Q_i on both sides of (3), and then we have

$$\begin{aligned} \Delta(Q_i) = & D_i - Q_i \mathbb{E}\{\beta_i - A_i | Q_i\} - Q_i \mathbb{E}\{C_i | Q_i\} + \\ & \mathbb{E}\{\beta_i C_i | Q_i\}, \end{aligned} \tag{4}$$

where $D_i = \frac{1}{2}\mathbb{E}\{\beta_i^2 + C_i^2 + A_i^2 | Q_i\}$.

By adding a penalty, the Lyapunov framework is able to minimize the objective specified in the penalty while keeping the queue length finite. Since preprocessing lowers the resolution of unit videos, we aim at minimizing the reduction in average resolution due to preprocessing while keeping the average delay low. Hence, we add a weighted conditional expectation of reduced resolution to both sides of (4), which now becomes

$$\begin{aligned} \Delta(Q_i) + V\mathbb{E}\{\phi_i | Q_i\} \leq & D_i - Q_i \mathbb{E}\{\beta_i - A_i | Q_i\} \\ & - \mathbb{E}\{(Q_i - \beta_i)\mathbb{E}\{C_i | \sigma, \beta_i, \phi_i\} - V\phi_i | Q_i\}, \end{aligned} \tag{5}$$

where the weight V is a predefined parameter. Based on the property of Lyapunov optimization framework, V can affect the tradeoff between the length of the transmission queue Q and the reduced resolution ϕ.

To minimize the the left-hand-side of (5), it is equivalent of maximizing the negative terms on the right-hand-side. Since we can only manipulate the preprocessing method σ, we attempt to maximize the following:

$$\mathbb{E}\{(Q_i - \beta_i)\mathbb{E}\{C_i|\sigma, \beta_i, \phi_i\} - V\phi_i|Q_i\}.$$

Therefore, the suggested preprocessing method σ_i in the timeslot (t_i, t_{i+1}) can be obtained as follows:

$$\sigma_i = \arg\max_{\sigma\in\Omega} \big\{(Q_i - \beta_i)\mathbb{E}\{C_i|\sigma, \beta_i, \phi_i\} - V\phi_i|Q_i\big\},$$

where Ω is the set of preprocessing methods that can be used on unit videos, which includes a preprocessing method that does nothing.

2) Video Selection: Once a preprocessing method is suggested, we adopt a deadline-based selection criteria to decide which unit video in the transmission queue gets checked first. Normally, unit videos that arrive earliest are checked first. The checking procedures include two steps. In Step 1, qualified unit videos must have the required resolution as specified by the suggested preprocessing method. In Step 2, the estimated preprocessing time on chosen unit videos should be less than the estimated transmission time of unit videos ahead of them in the queue.

It is likely that the first suggested preprocessing method fails to find a unit video of required resolution in the transmission queue. In such a case, the next suggested preprocessing method is considered, and will be chosen if a unit video of requirement resolution is found in the transmission queue. If the next suggested is the preprocessing method that does nothing, then it is confirmed that no preprocessing is applied for the timeslot concerned.

IV. Real-world Experiments

A. Implementation

To validate our proposed approaches, we implemented our resolution-maximization algorithm and the Lyapunov based video preprocessing algorithm on the Android platform. We adopted an Android developer phone (Nexus 5) as the wearable camera, since it allows us to specify which video encoding profile to use in video recording. We adopted another Android developer phone (Nexus 6) as the smartphone that is responsible for forwarding and, if needed, preprocessing videos. Nexus 5 was configured to connect to Nexus 6 via the Wi-Fi access point provided by the latter. Nexus 6 was equipped with cellular interface that supports 2G, 3G and 4G networks. We used a quad-core workstation that runs Windows Server 2012 to host uploaded video files. It supports 100 Mbits/second wired network connection.

To enable HTTP live broadcasting on Nexus 5, we used a third-party library named *Android-eye* [8] that can generate `.M3U8` playlists for subscribed users to download the video files specified in the playlists. Videos were transmitted from Nexus 5 to Nexus 6 via Wi-Fi, and were temporarily stored in a transmission queue (device memory or SD card). On Nexus 6, videos in the transmission queue were not eliminated from the queue until being successfully received by the workstation. To preprocess videos on Nexus 6, we used another third-party library named *Android-transcoder* [9] that utilizes hardware acceleration to boost video transcoding. On the workstation, we wrote some PHP programs that receive uploaded video files, and send back the finishing time of video uploading.

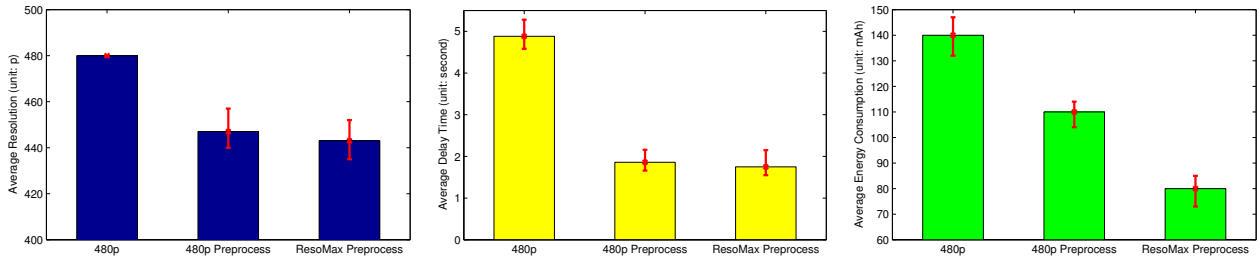
We monitored the realtime cellular bandwidth by tracking the amount of transmitted bytes and the actual transmission time for every interval of time. We run a background thread on Nexus 6 to periodically generate the video resolution adjusting decisions and the preprocessing decisions based on specific algorithms. The video resolution adjusting decision was sent from Nexus 6 to Nexus 5 as a HTTP request, which, if different from previous one, resulted in the change of video recording resolution on Nexus 5.

B. Experiment Setup & Results

We compared the performance of three approaches, namely the fixed 480p approach, the fixed 480p approach that adopts preprocessing, and our approach. In the experiments, we walked around our campus while carrying both Nexus 5 and Nexus 6, and enabled one approach for each live broadcast session (120 seconds). The initialization delay was set to be 20 seconds, the average delay constraint was set to be 2 seconds, and the preprocessing parameter V was set to be 10,000. Each approach was evaluated on multiple sessions, and results were averaged as shown in Figure 3.

Figure 3a shows that the fixed 480p approach achieves the highest average resolution (480p), while the other two approaches only achieve an average resolution of around 450p due to the adoption of video preprocessing. This suggests that the network conditions could not smoothly upload videos of 480p. On the other hand, Figure 3b shows that the fixed 480p approach incurs an average delay time of around 4.9 seconds, whereas the other two approaches only incur an average delay time of around 1.9 seconds. Although the adoption of video processing reduces the achieved average resolution, it shortens the average delay time. By selecting another value for the preprocessing parameter V, we can further tune the tradeoff between the average resolution and the average delay time.

In order to demonstrate the benefits of our resolution-maximizing algorithm, we also measured the energy consumption of each approach on Nexus 6 (as shown in Figure 3c) via a battery statistics tool provided by Android. The main sources of energy consumption come from the data transmission and, if any, video preprocessing. Comparing the 480p approach and the 480p approach with preprocessing, we observe that video preprocessing can effectively reduce the energy consumption of data transmission by reducing the amount of transmitted data. Furthermore, our approach incurs even lower energy consumption on Nexus 6, because its resolution-maximization algorithm adjusts the video recording resolution based on



(a) The Average Resolution (b) The Average Delay Time (c) Energy Consumption on Nexus 6

Figure 3: Real-world Experiment Results

monitored network bandwidths, which reduces the invocation times of video preprocessing.

V. Related Work

The HTTP Live Streaming (HLS) protocol [2] has become the de-facto solution for video streaming on mobile devices. Much work [10–12] has been done to improve the performance of this protocol. In [10], adynamic scheduling methodology is proposed to enable fast video transcoding for MPEG DASH (Dynamic Adaptive Streaming over HTTP) in a cloud environment. To avoid wasting transcoding resources, the authors propose several online transcoding policies that only segments that are actually requested are transcoded [11]. However, these techniques cannot be directly used on live broadcast from wearable cameras, since these devices (including smartphones) do not provide enough computing power to transcode videos to various dimensions in realtime.

In the area of client-to-server video upload, a more widely studied topic is about the uploading of stored videos. Movisode [13] is an event-centric system that supports on-demand uploading of videos from multiple mobile devices based on specific user queries. In Movisode, however, videos are already recorded and stored on mobile devices, and no preprocessing methods are done before they are uploaded. QuiltView [14] is a crowdsourced system based on wearable devices that sends back live video snippets of surroundings upon other users' queries. It emphasizes the searchability of video contents based on their geolocation, instead of optimizing the video transmission. Our work emphasizes the timely delivery of recorded videos, and adopts dynamic resolution adjustment and video preprocessing to counteract the adverse effect of suddenly worsened network conditions.

VI. Conclusion

Live broadcast from wearable cameras facilitates the sharing of individuals' views of the world, and allows people to gain similar experience without actually being present. However, constantly changing network conditions pose a great challenge to smooth live broadcast in places where Wi-Fi is unavailable. Our proposed dynamic coding approach that utilizes resolution adjust on wearable cameras and video preprocessing on smartphones has proved its effectiveness in improving the smoothness of live broadcast under cellular networks. Our experiment results show that our proposed resolution-maximizing algorithm can substantially lower the overheads of video preprocessing on smartphones, whereas our proposed Lyapunov based video proprocessing algorithm can greatly reduce the average delay time for live broadcast, at the cost of slightly compromised video quality. Our future work will be focused on extending the application of our proposed approach to other areas, such as realtime object detection and identification from wearable cameras.

Acknowledgement

This work is supported by HK PolyU G-YBJV and G-YBAD.

References

[1] The rise of GoPro: why wearable cameras make us film everything. [Online]. Available: http://goo.gl/eKWZrH
[2] R. Pantos, "HTTP Live Streaming," 2015.
[3] Broadcast live from your GoPro Hero®. [Online]. Available: http://goo.gl/e6X7w4
[4] O. Oyman and S. Singh, "Quality of Experience for Http Adaptive Streaming Services," *Communications Magazine, IEEE*, vol. 50, no. 4, pp. 20–27, 2012.
[5] J. Li, K. Bu, X. Liu, and B. Xiao, "ENDA: Embracing Network Inconsistency for Dynamic Application Offloading in Mobile Cloud Computing," in *Proceedings of the second ACM SIGCOMM workshop on Mobile cloud computing*. ACM, 2013, pp. 39–44.
[6] J. Li, Z. Peng, B. Xiao, and Y. Hua, "Make Smartphones Last A Day: Pre-Processing Based Computer Vision Application Offloading," in *Proceedings of the 12th IEEE SECON*, June 2015, pp. 462–470.
[7] M. J. Neely, *Stochastic Network Optimization with Application to Communication and Queueing Systems*. Morgan and Claypool Publishers, 2010.
[8] Android Eye. [Online]. Available: https://github.com/Teaonly/android-eye
[9] Android Transcoder. [Online]. Available: https://github.com/ypresto/android-transcoder
[10] H. Ma, B. Seo, and R. Zimmermann, "Dynamic Scheduling on Video Transcoding for Mpeg Dash in the Cloud Environment," in *Proceedings of the 5th ACM Multimedia Systems Conference*. ACM, 2014.
[11] D. K. Krishnappa, M. Zink, and R. K. Sitaraman, "Optimizing the Video Transcoding Workflow in Content Delivery Networks," in *Proceedings of the 6th ACM Multimedia Systems Conference*, 2015, pp. 37–48.
[12] F. Chen, C. Zhang, F. Wang, J. Liu, X. Wang, and Y. Liu, "Cloud-Assisted Live Streaming for Crowdsourced Multimedia Content," *Multimedia, IEEE Transactions on*, vol. 17, no. 9, pp. 1471–1483, 2015.
[13] S. P. Venkatagiri, M. C. Chan, W. T. Ooi, and J. H. Chiam, "On Demand Retrieval of Crowdsourced Mobile Video," *Sensors Journal, IEEE*, vol. 15, no. 5, pp. 2632–2642, 2015.
[14] Z. Chen, W. Hu, K. Ha, J. Harkes, B. Gilbert, J. Hong, A. Smailagic, D. Siewiorek, and M. Satyanarayanan, "QuiltView: a Crowd-Sourced Video Response System," in *Proceedings of the 15th Workshop on Mobile Computing Systems and Applications*. ACM, 2014, p. 13.