Improving Power Efficiency for Online Video Streaming Service: A Self-Adaptive Approach

Jingyu Zhang, Zhi-Jie Wang, Kun Wang, Song Guo, *Senior Member, IEEE,* Bin Wang, and Minyi Guo, *Senior Member, IEEE*

Abstract—The video streaming technique and 4G LTE networks have been developing at a high speed, while the advancement of the battery technology is relatively slow. A lot of efforts have been made on understanding the power consumption in general 4G LTE networks, while few attention focused on reducing the power consumption of mobile devices for online video streaming in 4G LTE networks. In this paper, we attempt to develop efficient optimization techniques to address this problem. It is an important and also interesting problem, while there are many challenges needing to be alleviated (e.g., the uncertainty of users' behavior modes). To attack the challenges, we suggest a self-adaptive method that can allow us to adjust various parameters dynamically and efficiently, achieving a relatively small energy consumption. We give the rigorous theoretical analysis for the proposed method, and conduct the empirical study to validate its effectiveness.

Index Terms—Video streaming, LTE networks, Power consumption.

1 Introduction

Video streaming, as a popular technology used for online video playing back, has been received much attention in mobile industry [1], [2], [3], [4], [5], and has already been applied into 4G LTE networks [6], [7], [8]. Recently, both the video streaming technique and 4G LTE networks have been developing at an amazing speed [2], [9], [10]. Yet, the advancement of the battery technology is relatively slow [11], [12]. It is particularly important to reduce the power consumption of mobile devices for online video streaming in 4G LTE networks [6], [8], [13], [14].

In existing literature, there are many works studying: (i) Power consumption related to online video streaming (see e.g., [6], [15], [16], [17], [18], [19], [20], [21]); however, these works *did not* cover the *LTE network environments*, consider the impact of *radio resource control* (RRC) tails, and address the challenge related to *users' behavior modes*. (ii) Power consumption in wireless networks (see e.g., [22], [23], [24], [25]); most of these works focused on energy consumption of *sensors*, and assumed to be in the traditional wireless networks environment. And (iii) power consumption in 4G LTE networks (see e.g., [26], [27], [28], [29], [30]); these works mainly focused on power consumption in the general LTE networks, the specific online video scenarios are not discussed. On the other hand, while the previous works [13], [29] based on the real experimental platform showed that, the saving room in the network part is large, and the number of RRC tails could be promising for optimizing the power consumption in such scenarios; yet, in those articles no targeted algorithms and/or techniques were proposed to optimize the power efficiency. The above facts naturally motivate this work. Specifically, in this paper we attempt to develop efficient strategies to address the problem above.

There are several main challenges needing to be alleviated: (i) the *video buffer* is usually employed for video segments downloading and storage, in order to avoid the video playing delay [7], [8]. Intuitively, one can use a larger buffer, it shall immediately bring us the benefit in terms of energy consumption. Yet, imagine if the behavior of a user u is in the *"unstable" mode*[1], video segments downloaded in the buffer prefetching phase are usually being cleaned without watching them; this shall incur some energy waste. Then, how to control the buffer size efficiently? (ii) the previous works [13], [29] have shown that the number of RRC tails could be promising for optimizing the power consumption. An immediate idea is to merge multiple *segment fetching sessions*[2] together. Essentially, existing techniques (see e.g., [8]) can be directly used to merge the segment fetching sessions. However, these methods ignore users' behaviors. That is, the "merged" video segments could be discarded without watching them, incurring also the energy waste. Then, how to merge the video segment fetching sessions wisely? (iii) one could argue that existing works (see e.g., [6]) have suggested different methods for different users'

- *J. Zhang and M. Guo are with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China. E-mail: zhangzhang@sjtu.edu.cn; guo-my@cs.sjtu.edu.cn*
- *Z.-J. Wang is with the Guangdong Key Laboratory of Big Data Analysis and Processing, School of Data and Computer Science, Sun Yat-Sen University, Guangzhou, China. E-mail: wangzhij5@mail.sysu.edu.cn*
- *K. Wang is with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China; and the Jiangsu High Technology Research Key Laboratory for Wireless Sensor Networks, Nanjing University of Posts and Telecommunications, Nanjing, China. E-mail: kun.wang1981@gmail.com*
- *S. Guo is with the Department of Computing, The Hong Kong Polytechnic University, Kowloon, Hong Kong. E-mail: song.guo@polyu.edu.hk*
- *B. Wang is with the School of Electronics, Electrical Engineering and Computer Science, Queen's University Belfast, United Kingdom. E-mail: B.Wang@qub.ac.uk*

1. Simply speaking, this mode refers to that, user tends to drag the process bar of the player. A more detailed explanation shall be covered in Section 2.

2. When a user enjoys the online video streaming service, the video streaming player downloads the video segments periodically. This implies that many *downloading sessions* shall be involved in this process; and usually the *downloading session* is called *segment fetching session.*

TABLE 1
Main symbols

Notation	Meaning
τ_s, τ_i, τ_{sd}	total service time of a segment, idle period with no traffic, playing back time duration of a video segment
τ_{rt}, τ_{ri}	RRC tail and idle time durations, respectively
f_s	file size of a video segment
$\overline{v}_p$, $\overline{v}_f$	average download speed in the buffer prefetching and feeding phases, respectively
ζ, ζ_o, $\zeta_{\tau_{(\cdot)}}$	total energy consumption for network activities, average power consumption for offline playing, power consumption during $\tau_{(\cdot)}$, respectively
u_t	continuous online video watching time
Δ^*	the size of each buffer adjustment
b_m	maximum number of video segments contained in b
b_t, Ω, Θ	thresholds used to trigger video segment fetching session, differentiate warm-up stage, trigger the adjustment of b_s
$\top$	upper bound of the number of video segments in a single segment fetching session

behavior modes. However, the users' behavior modes are assumed to be available beforehand in their work. In the usual case, we could not know users' behavior modes beforehand. Then, how to reduce the power consumption when users' behavior modes are unavailable beforehand?

To attack these challenges, this paper suggests a self-adaptive method, which is easy-to-understand and also easy-to-deploy. our method can be applied in the client-side directly, and thus does not involve many complicated operations in the server-side. The central idea of our method is to exploit the continuous video watching time to predict users' behavior modes, and then dynamically adjust corresponding parameters. We give the rigorous theoretical analysis for our proposed method. The analysis verifies the feasibility and effectiveness of our method, from the theoretical perspective. We also conduct empirical study to demonstrate the effectiveness of our proposed method. The experimental results show that the proposed method consistently outperforms the classical method as well as other competitors adapted from existing methods.

The remainder of this paper is organized as follows. Section 2 formally describes our problem. Section 3 presents our proposed method and Section 4 gives the rigorous theoretical analysis. Experimental results are covered in Section 5, and we conclude this paper in Section 6.

2 Problem Definition

In this section we start by introducing some preliminaries, and then formulate the settings of our problem, and finally present the formal definition of our problem. For ease of reference, Table 1 summarizes the main symbols used in the paper.

Regarding the online video streaming, a widely used *timing scheme* involves two levels [13], [27], [29], [31]: (i) the video streaming traffic level, see the bottom of Figure 1. At this level, there is an interactive process between the mobile client and the server. (ii) the RRC level, see the top of Figure 1. At this level, after finishing the segment transmission, even if no new data transmission happens, the mobile client still stays at the RRC connected state within a time duration (before entering the RRC idle state mode); this time duration is called the *RRC tail*. Usually, the power in

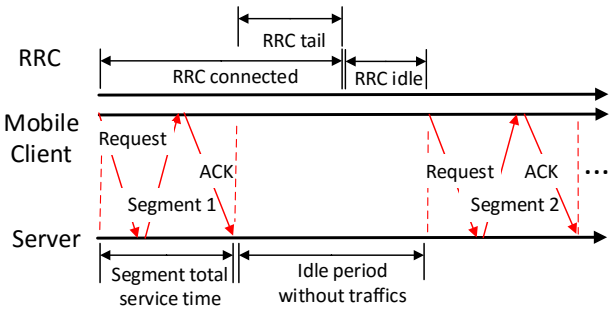

Fig. 1. Timing scheme of video streaming.

the RRC connected state is more than that in RRC idle state [29], [31]. For brevity, we denote by τ_s, τ_i, and τ_{sd} the total service time of a segment, the idle period with no traffic, and the playing back time duration of a video segment, respectively. Let τ_{rt} and τ_{ri} be the RRC tail time duration and the RRC idle time duration, respectively.

Problem settings. Given a 4G LTE network $\mathbb{N}$, assume that a user u is with a mobile device d (e.g., smartphone), on which a video player p is deployed. Let n be the number of total video segments, and f_s be the file size of a *video segment*. Without loss of generality, assume that the maximum number of video segments contained in the buffer is b_m, and the buffer threshold used to trigger another *segment fetching session* is b_t. When one plays back an online video, the video may last for a relative long time. In this paper, the early stage of the video being played back is referred to as the *warm-up stage*.

For the user u, assume that one of two behavior modes could be involved. (1) *Stable mode*: the user neither skips to another part of the current video, nor quits the current video. In other words, the user is likely to watch video for a relative long time. (2) *Unstable mode*: the user tends to *skip* to another part of the current video, or quits the current video playing even if the current video does not finish; once the "skip" happens, the buffer shall clean the current buffered video segments and restart. For example, a user is watching the video at 00:05, he may skip to the video at 00:15 if he conceives the video near at 00:05 uninteresting. In this case, the buffer shall clean the buffered content, and shall fetch new video segments to fill itself.

Note that, in the entire video playing process, the video buffer b involves two phases. (1) *Buffer prefetching*: a set of consecutive video segments are prefetched and filled into b. For convenience, denote by τ_{bp} the buffer prefetching time. (2) *Buffer feeding*: the mobile client requests the video segments, based on the specific demand. Without loss of generality, let $\overline{v}_p$ and $\overline{v}_f$ be the average download speed in the buffer prefetching phase and in the buffer feeding phase, respectively.

Problem statement. Let ζ be the total energy consumption of the mobile device for network activities, ζ_o be the average power consumption when playing back the video *offline*, and $\zeta_{\tau_{(\cdot)}}$ be the power consumption during $\tau_{(\cdot)}$ (e.g., $\zeta_{\tau_{ri}}$ denotes the average power consumption during the RRC idle τ_{ri}), respectively. As mentioned earlier, a user u could be in one of two behavior modes. For these two modes, the energy consumption are different. Observe that, in the

usual case u's behavior mode is unknown beforehand. In what follows, we first formulate the energy consumption for these two modes, and then state our problem formally.

First, consider the *stable mode*, its power consumption ζ consists of two parts: (1) the power consumption in the buffer prefetching phase, i.e., $\frac{b_m}{\overline{v}_p}\zeta_{\tau_{bp}}$; and (2) the power consumption in the buffer feeding phase, i.e., $(\frac{n-b_m}{b_m-b_t}) \times (\frac{f_s}{\overline{v}_f} \times (\zeta_{\tau_s}-\zeta_o) + (\tau_{sd}-\frac{f_s}{\overline{v}_f}) \times (\zeta_{\tau_i}-\zeta_o))$. Notice that, for (2) it essentially has two sub-parts: (a) the energy consumption of network transmission (see the part before the "+"); and (b) the energy consumption of network idle period (see the part after the "+").

Observe that $\tau_i = \tau_{rt} + \tau_{ri}$ (c.f., Figure 1). Thus, based on the above analysis one can obtain the following:

$$\begin{aligned}\zeta =& \frac{b_m}{\overline{v}_p}\zeta_{\tau_{bp}} + \left(\frac{n-b_m}{b_m-b_t}\right) \times \left(\frac{f_s}{\overline{v}_f}(\zeta_{\tau_s}-\zeta_o) + \tau_{rt}(\zeta_{\tau_{rt}}-\zeta_o)\right. \\ &\left. + (\tau_{sd} - \frac{f_s}{\overline{v}_f} - \tau_{rt})(\zeta_{\tau_{ri}}-\zeta_o)\right)\end{aligned} \tag{1}$$

We proceed to consider the *unstable mode*. Assume, without loss of generality, that the number of skips is $i-1$; naturally, video playing shall be divided into i parts. Denote by n_i the number of video segments in the ith part of video playing. Then, the power consumption in the buffer prefetching phase is $i \times \frac{b_m}{\overline{v}_p}\zeta_{\tau_{bp}}$, and the power consumption in the buffer feeding phase is $\frac{(n_1+\cdots+n_i-b_m\times i)}{b_m-b_t}(\frac{f_s}{\overline{v}_f}(\zeta_{\tau_s}-\zeta_o) + (\tau_{sd}-\frac{f_s}{\overline{v}_f})(\zeta_{\tau_i}-\zeta_o))$. Similarly, one can obtain the following (since $\tau_i = \tau_{rt}+\tau_{ri}$, recall Figure 1):

$$\begin{aligned}\zeta =& i \times \frac{b_m}{\overline{v}_p}\zeta_{\tau_{bp}} + \frac{(n_1+\cdots+n_i-b_m\times i)}{b_m-b_t}\left(\frac{f_s}{\overline{v}_f}(\zeta_{\tau_s}-\zeta_o)\right. \\ &\left. + \tau_{rt}(\zeta_{\tau_{rt}}-\zeta_o) + (\tau_{sd}-\frac{f_s}{\overline{v}_f}-\tau_{rt})(\zeta_{\tau_{ri}}-\zeta_o)\right)\end{aligned} \tag{2}$$

In summary, our goal is to obtain a small value in terms of ζ, while considering the challenges mentioned earlier.

3 Self-adaptive solution

To address the challenges mentioned earlier, we propose a self-adaptive solution for saving the power consumption. Our method provides a unified framework for different behavior modes; also, it can adjust the buffer size and merge video segment fetching sessions dynamically and flexibly. The difficulty to design this method is to "balance" various parameters efficiently. Generally speaking, our method consists of two main steps. First, a set of parameters are initialized and then it prefetches a set of video segments to fill the buffer. Our key contribution in the first step is the design of various parameters; these parameters together with our strategies (developed in the second step) shall collaboratively contribute to the reduction of energy consumption. Second, we dynamically adjust the parameters based on our proposed strategies. Our strategies shall guide us to fetch remaining video segments flexibly and wisely in the video segment feeding phase. In what follows, we examine the details of our solution.

One of important elements is b_t, which serves as the threshold to trigger video segment fetching (recall Section 2). For the conventional online video player, if the buffer b is not full, another video segment fetching session shall be triggered. That is, the default value of b_t is equal to $b_m - 1$, where b_m refers to the maximum number of video segments contained in b (recall Section 2). Instead, in our design we adjust b_t dynamically. Let $\mho$ ($\in [1, +\infty)$) be a predefined threshold used to differentiate the *warm-up stage*[3]. We use a "diminishing" policy for adjusting the threshold b_t. In other words, the longer the continuous online video watching time u_t is, the smaller b_t will be set. This implies that, in the rest of segment fetching sessions, more video segments shall be fetched/downloaded. Specifically, our diminishing policy is as follows.

$$b_t = \begin{cases} b_t - \left\lfloor \frac{\frac{u_t}{\tau_{sd}} - X}{(b_m-b_t)\times 2} \right\rfloor, & if\ (b_m - b_t) < \mho \\ b_t - 1, & otherwise \end{cases} \tag{3}$$

where $\frac{u_t}{\tau_{sd}}$ is the total number of continuously viewed video segments, $(b_m-b_t)\times 2$ implies we use two segment fetching sessions as the evaluation unit in the warm-up stage. X denotes the number of viewed video segments before the latest change on b_t, and it is equal to $(1+2+\cdots+(b_m-b_t-1))\times 2$, i.e., $(b_m-b_t)(b_m-b_t-1)$.

In summary, Eq. 3 reflects that, if $b_m - b_t < \mho$, it is in the warm-up stage. This implies that it could be hard to predict the user behavior. In this case, b_t is decreased after more segment fetching sessions are finished. In contrast, if $b_m - b_t \geq \mho$, the possibility to be in stable mode is increasing gradually. In this case, b_t is decreased by one after one segment fetching session is finished.

Note that, there is an issue needing to be emphasized. That is, when b_t is smaller and smaller, more and more video segments are fetched in a single segment fetching session. In this case, if users skip to another part of the video. This shall incur the significant waste, regardless of energy or bandwidth. To alleviate this issue, we introduce a parameter $\top$, which represents the upper bound of the number of video segments in a single segment fetching session. In other words, when $b_m - b_t = \top$, we shall not further use the "diminishing" policy.

Another important element is the buffer size b_s. For the conventional online video player, it is usually set to a fixed value (e.g., 10 or 20 seconds). Imagine if the number of video segments in a single segment fetching session is larger than $b_m - b_t$. In this case, the system could not work well. To alleviate this dilemma, in our design we adjust b_s self-adaptively. Let Θ ($\in (0,1)$) be a threshold used to trigger the adjustment of b_s, and Δ^* be the size of each adjustment. Specifically, when $\frac{b_t}{b_m} \leq \Theta$, we set $b_s = b_s + \Delta^*$.

Note that, when we enlarge b_s, b_m (i.e., the maximum number of video segments contained in b) is naturally enlarged. In this case, the number of segments in the subsequent segment fetching session could sharply increase, incurring negative results such as the waste of bandwidth and energy (the reason is similar to our previous analysis). We remedy this trouble by enlarging the value of b_t. Specifically, only if b_s is set to $b_s + \Delta^*$, we immediately set $b_t = b_t + \Delta$, where $\Delta = \Delta^*/\tau_{sd}$.

3. Note that, in this stage the number of video segments in a single segment fetching session is relatively small.

Algorithm 1 Self-adaptive method

```
1:  initialize parameters
2:  prefetch b_m video segments, and set n_c = b_m
3:  while (n_c ≠ 0)
4:    if (a video segment has been played back)
5:      set n_c = n_c − 1, u_t = u_t + τ_sd
6:    if (n_c <= b_t )
7:      if (the remaining video segments are null)
8:        continue; // go to Line 3
9:      if (b_t / b_m ≤ Θ) //need to adjust buffer size
10:       set b_s = b_s + Δ*, b_t = b_t + Δ, b_m = b_m + Δ, and
            fetch Δ + (b_m − b_t) video segments
11:     else // do not need to adjust buffer size
12:       fetch b_m − b_t video segments
13:   if ( b_m − b_t < ⊤ ) // update b_t
14:     if (b_m − b_t < ʊ) // warm-up stage
15:       update b_t based on the 1st item in Eq. 3
16:     else //warm-up is finished
17:       update b_t based on the 2nd item in Eq. 3
```

The pseudo-codes of our method are shown in Algorithm 1. In general, Lines 1-2 serve as the initialization of parameters, and the prefetching of video segments. Lines 3-17 are used to process the buffer feeding phase. More specifically, once a video segment has been played back, we remove it from the buffer, and accumulate the continuous watching time u_t (Lines 4-5). When the number of buffered video segments n_c is smaller than the threshold b_t, we start a new segment fetching session (Lines 6-12). Finally, Lines 13-17 are used to update b_t based on Eq. 3. We remark that, if the user skips to another part of the video or starts a new video, the algorithm restarts the above process.

4 Theoretical Analysis

In this section, we examine the effectiveness of our proposed method, from theoretical perspective. As mentioned earlier, our method works for both stable and unstable modes. For ease of presentation, we first analyze the stable mode, followed by the unstable mode.

Stable mode. Let b'_m be the initial maximum number of video segments contained in b. (Here the term "initial" maximum number refers to the value before enlarging the buffer.) Denote by N_m the number of downloaded video segments in the last segment fetching session. It equals $(n - b'_m - (\Delta \times n_e) - 2(\mho - 1) - (\top - \mho - 1)) \mod \top$, where "$2(\mho - 1)$" denotes the total downloaded video segments before $(b_m - b_t)$ reaches $\mho$, and "$(\top - \mho - 1)$" denotes the total downloaded video segments when $(b_m - b_t)$ is between $\mho$ and $\top$. That is, $N_m = (n - b'_m - (\Delta \times n_e) - \mho - \top + 3) \mod \top$. In addition, let N be the number of video segment fetching sessions after $(b_m - b_t)$ reaches $\top$, it is computed as $N = \left\lfloor \frac{n - b'_m - (\Delta \times n_e) - \mho - \top + 3}{\top} \right\rfloor$.

Let $\overline{v}_f^k$ be the average download speed for the fetching session that fetches k segments, and ζ_f^k be the power consumption paid for such a fetching session. Similar to Eq. 1, one can compute ζ_f^k as $\frac{f_s \times k}{\overline{v}_f^k}(\zeta_{\tau_s} - \zeta_o) + \tau_{rt}(\zeta_{\tau_{rt}} - \zeta_o) + (\tau_{sd} - \frac{f_s \times k}{\overline{v}_f^k} - \tau_{rt})(\zeta_{\tau_{ri}} - \zeta_o)$.

Let n_e be the total times of enlarging buffer in the buffer feeding phase, and ζ_a be the additional energy paid for each buffer enlarging process. Based on the equations above, we can compute the "improved" energy consumption in the stable mode as follows.

$$\begin{aligned}\zeta = &\frac{b'_m}{\overline{v}_p} \times \zeta_{\tau_{bp}} + (n_e \times \zeta_a) + 2 \times \sum_{k=1}^{\mho-1} \zeta_f^k \\ &+ \sum_{k=\mho}^{\top-1} \zeta_f^k + (N \times \zeta_f^{\top}) + \zeta_f^{N_m}\end{aligned} \tag{4}$$

In the above equation, the part before the second "+" refers to the power consumption in the buffer prefetching phase and buffer enlarging process. Note that, both these phases have power-efficient traffic, and the more the segments to be downloaded are, the more power-efficient the traffic is [32], [33]. It is not hard to realize that this part corresponds to the part before the first "+" in Eq. 1. One can observe that the number of segments contained in this part (c.f., Eq. 4) is $(b'_m + n_e \times \Delta)$. It is larger than b_m in Eq. 1 (notice: here b_m is essentially equal to b'_m). So, our method is more power-efficient in this part.

We next consider the part after the second "+" in Eq. 4. This part refers to the power consumption in the buffer feeding phase, which could produce the RRC tails. Note that, the smaller the number of RRC tails is, the fewer the energy shall be consumed [13]. It is not hard to see that this part corresponds to the part after the first "+" in Eq. 1. One can observe that the number of RRC tails in this part (c.f., Eq. 4) is $(\mho - 1) \times 2 + (\top - 1) + N + 1$, i.e., $(\mho + \top + N - 2)$. It is smaller than $\frac{n - b_m}{b_m - b_t} = (n - b_m)$ in Eq. 1[4]. This is because the proposed method merges many single segment fetching sessions together. On the other hand, when many single segment fetching sessions are merged, the segments to be downloaded in a single session turns more, this shall achieve more power-efficient traffic (as we discussed earlier). Combining these reasons, our method is also more power-efficient in this part. To summarize, our proposed method achieves the optimization in the stable mode.

Unstable mode. Recall Section 2, the total number of user skips is assumed to be $(i - 1)$. Without loss of generality, assume that the buffer size b_s will be enlarged n_e^i times for the ith video playing part. Denote by N_m^i the number of downloaded video segments in the last fetching session, in terms of the ith video playing part. Then, N_m^i is computed as $(n_i - b'_m - (\Delta \times n_e^i) - 2(\mho - 1) - (\top - \mho - 1)) \mod \top$. That is, $N_m^i = (n_i - b'_m - (\Delta \times n_e^i) - \mho - \top + 3) \mod \top$.

Denote by N^i the number of video segment fetching sessions after $(b_m - b_t)$ reaches $\top$, in terms of the ith video playing part. Then, it is computed as $N^i = \left\lfloor \frac{n_i - b'_m - (\Delta \times n_e^i) - \mho - \top + 3}{\top} \right\rfloor$.

Based on the equations above, we can compute the "improved" energy consumption in the unstable mode as follows.

$$\begin{aligned}\zeta = &\sum_{k=1}^{i} \left(\frac{b'_m}{\overline{v}_p} \times \zeta_{\tau_{bp}} + n_e^i \times \zeta_a \right) + \sum_{k=1}^{i} \Bigg(2 \times \sum_{k=1}^{\mho-1} \zeta_f^k \\ &+ \sum_{k=\mho}^{\top-1} \zeta_f^k + (N^i \times \zeta_f^{\top}) + \zeta_f^{N_m^i} \Bigg)\end{aligned} \tag{5}$$

4. Remark that, in the traditional design, $b_m - b_t = 1$, and so $\frac{n - b_m}{b_m - b_t} = (n - b_m)$.

As similar as Eq. 4, one can consider two parts in Eq. 5 based on the second "+". For the first part, in each single segment fetching session, the number of segments contained in the session is $(b'_m + n^i_e \times \Delta)$, larger than b_m in Eq. 2. Thus, our method is more power-efficient in this part (the reason is same to our previous analysis). For the second part, in each single segment fetching session the total number of RRC tails in the buffer feeding phase is $(\mho - 1) \times 2 + (\top - 1) + N^i + 1 = (\mho + \top + N^i - 2)$, which is smaller than the corresponding number $\frac{n_i - b_m}{b_m - b_t} = (n_i - b_m)$ in Eq. 2. Thus, our method is also more power-efficient in the latter part (again, the reason is same to our previous analysis). To summarize, in the unstable mode our solution also achieves the optimization.

5 Empirical Study

In this section, we experimentally evaluate our proposed method (known as SA, for short). The platform used in our experiments is composed of three major components: (i) Samsung Galaxy S5, one of the most popular smartphones; (ii) a professional *mobile power meter* — Monsoon power monitor [34]; and (iii) DASH-IF player, the official DASH industry forum reference and production video streaming player [35]. Besides, some other auxiliary components (including *wireshark* [36], *QXDM* [37]) are also used in our experiments. Here Wireshark is used for the traffic information collection (e.g., TCP packet transmission data and HTTP messages), and QXDM is used for the RRC state information collection. In addition, the video server is built with Jetty [38].

We compare our method against four competitors. (i) *Cls*: it is the most classic method for online video streaming [39]. (ii) *MF2*: it was mentioned in [8]; its basic idea is to merge two *segment fetching sessions* into one. This method can directly reduce the number of RRC tails. (iii) *MF4*: it is a variant of MF2; the main idea is to merge four segment fetching sessions into one [6]. (iv) *Opt*: it downloads all video segments in a single segment fetching session [13], and then plays back the video offline. This method obtains the minimum power consumption. To some extent, it can be also viewed as a variant of MF2. Essentially, it is more extreme in the segment fetching session, compared with MF4. Note that, this method is an "ideal" prototype, yet it is impractical in the real online video streaming application. Here it mainly serves as a reference.

We report the results of four sample videos: (i) SL180; (ii) SL360; (iii) SL720, and (iv) SL1080, respectively. The length of each video is 600 seconds, and each video contains 300 video segments (i.e., chunks). Each video segment lasts for about 2 seconds. The resolutions of these videos are 180p, 360p, 720p, and 1080p, respectively. These videos are available at http://www.digitalprimates.net. Remark that, we also tested other video datasets; the experimental results are similar with ours, and so we do not show them, in order to save space. The experimental parameters Δ, $\mho$, $\top$, and Θ can be adjusted manually. We test these parameters offline, and empirically set to 5, 4, 10 and 50% respectively, in order to achieve the good performance. In addition, the default value of b_s is set to 30 seconds. Note that, although our method can be applied to arbitrary bitrates in terms of video streaming, we here use the stable network environment and fixed bitrates, in order to make our experiments under control.

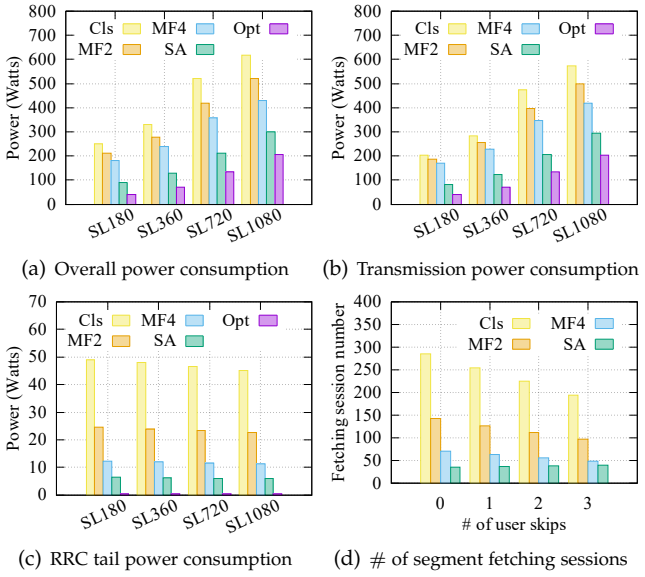

Fig. 2. Power consumption and number of fetching sessions. Note that, the "Opt" method mainly serves as a reference.

In our experiments, we test both the stable and unstable modes. When a user skips to watch another part of the video, we assume the "skip duration" is uniformly distributed from 20 to 50 seconds. (Here the skip duration refers to the step length of each skip, when a user drags the process bar.) The "zero skips" means there is no user skips during the video playing (i.e., in the stable mode). In what follows, we first cover the experimental results without user skips, followed by the experimental results with user skips.

Figure 2(a) shows the overall power consumption in the stable mode. We can see that (i) both MF2 and MF4 can improve the power efficiency, while they are inferior than SA; and (ii) SA is most close to Opt, and outperforms significantly Cls. These phenomena could be due to that our method can merge enough segment fetching sessions together, compared with the former three methods. Note that, no matter how long the user watches the video, the buffer parameters are constant for the former three methods. In contrast, SA collects the continuous video watching time, and adaptively adjust the buffer parameters (e.g., the threshold b_t, the buffer size b_s). As a result, the more video segment fetching sessions can be merged, saving the power consumption. Figure 2(d) (c.f., $\# = 0$) also provides us some evidences; in this figure the number of segment fetching sessions is plotted. Furthermore, Figure 2(b) and 2(c) show the power consumption for downloading the video segments (i.e., the transmission power consumption) and for the RRC tails, respectively. We can see that SA consumes less power than Cls, MF2 and MF4. This implies that SA can achieve more power-efficient traffic than other three methods.

Figure 3 shows the results in the unstable mode. Note that, in this mode Opt can not work; we here do not discuss it. We can see from this figure that, the more the number of user skips is, the more power consumption will be paid. This is because, if users skip to another part of the video,

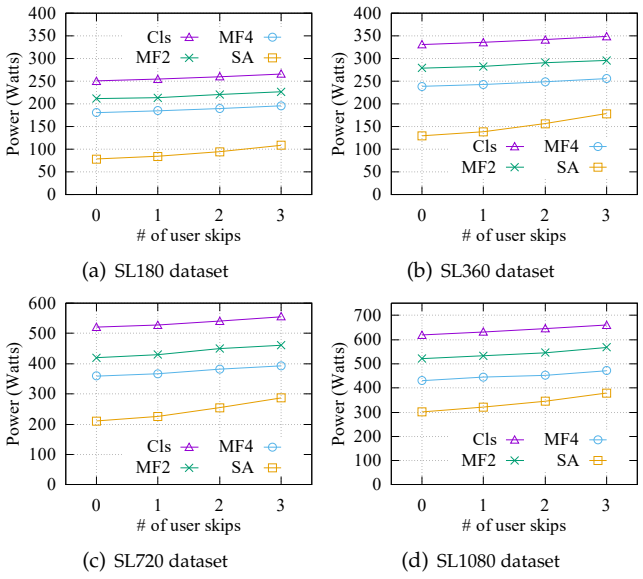

Fig. 3. Impact of user skips. Note that, it is essentially in the stable mode when the number of skips is 0.

(i) another additional buffer prefetching phase will be paid; and (ii) the buffered video segments will be removed from the buffer, without watching them (notice: they have already consumed additional power previously). On the other hand, we can see that, the power consumption of SA grows with the increase of user skips, and the growth speed is greater than Cls, MF2, and MF4. This is because, in the unstable mode SA can not stay in the most power-efficient stage for a long time. However, from the experimental results, we can still see that the overall performance of SA is still better than Cls, MF2, and MF4. This further demonstrates the effectiveness of our proposed approach. Again, we can see from Figure 2(d) (cf., $\# = 1, 2, 3$) that, SA can achieve a smaller value in terms of the number of fetching sessions, even if there are user skips. This essentially explain why SA outperforms the former three methods.

6 Conclusion

In this paper, we analyzed the major challenges to reduce the power consumption for online video streaming service; and proposed a self-adaptive method that addresses these challenges efficiently. The central idea of our method is to exploit the continuous video watching time to predict users' behavior modes, and then dynamically adjust different parameters, achieving a relatively small power consumption. We provided a rigorous theoretical analysis, and also experimentally validated the feasibility and effectiveness of the proposed method.

Acknowledgment

We would like to acknowledge the editors and anonymous reviewers for their instructive suggestions. Also, we gratefully acknowledge the warm help of Prof. Chunyi Peng and Prof. Sheng Wei, who have offered us valuable advices. This work was sponsored by the National Basic Research ("973") Program of China (2015CB352403), the National Natural Science Foundation of China (61261160502, 61272099 and 61572262), the Scientific Innovation Act of STCSM (13511504200), the EU FP7 CLIMBER project (PIRSES-GA-2012-318939), NSF of Jiangsu Province (BK20141427) and China Postdoctoral Science Special Foundation.

References

[1] Ericsson mobility report: on the pulse of the networked society. http://www.ericsson.com/mobility-report, 2016.
[2] Mohammad Reza Zakerinasab and Mea Wang. A cloud-assisted energy-efficient video streaming system for smartphones. In *IWQoS*, pages 1–10, 2013.
[3] Kun Wang, Jun Mi, Chenhan Xu, Qingquan Zhu, Lei Shu, and Der Jiunn Deng. Real-time load reduction in multimedia big data for mobile Internet. *ACM Trans. on Multimedia Computing Communications & Applications*, 12(5s):76:1–76:20, 2016.
[4] Wei Xiang, Gengkun Wang, Mark Pickering, and Yongbing Zhang. Big video data for light-field-based 3D telemedicine. *IEEE Network*, 30(3):30–38, 2016.
[5] Xiaoyan Guo, Yu Cao, and Jun Tao. SVIS: Large scale video data ingestion into big data platform. In *DASFAA Workshops*, pages 300–306, 2015.
[6] Wenjie Hu and Guohong Cao. Energy-aware video streaming on smartphones. In *INFOCOM*, pages 1185–1193, 2015.
[7] Xin Li, Mian Dong, Zhan Ma, and Felix C. A. Fernandes. Greentube: power optimization for mobile videostreaming via dynamic cache management. In *ACM Multimedia Conference*, pages 279–288, 2012.
[8] Sheng Wei, V. Swaminathan, and Mengbai Xiao. Power efficient mobile video streaming using HTTP/2 server push. In *MMSP*, pages 1–6, 2015.
[9] GSMA global mobile economy report 2016. http://gsmamobileeconomy.com/global/.
[10] Kun Wang, Yun Shao, Lei Shu, and Chunsheng Zhu. Mobile big data fault-tolerant processing for ehealth networks. *IEEE Network*, 30(1):36–42, 2016.
[11] Mike Williams. Why are mobile phone batteries still so crap. http://www.techradar.com/news/phone-and-communications/mobile-phones/why-are-mobile-phone-batteries-still-so-crap–1162779.
[12] Ekarat Rattagan, T. H. Chu, Ying Dar Lin, and Yuan Cheng Lai. SEMI: semi-online power estimates for smartphone hardware components. *T-SUSC*, 1(2):54–62, 2016.
[13] Jingyu Zhang, Gan Fang, Minyi Guo, and Chunyi Peng. How video streaming consumes power in 4G LTE networks. In *17th IEEE International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pages 1–3, 2016.
[14] Kun Wang and Yue Yu. A query-matching mechanism over out-of-order event stream in iot. *International Journal of Ad Hoc & Ubiquitous Computing*, 13(3/4):197–208, 2013.
[15] Yunmin Go, Oh Chan Kwon, and Hwangjun Song. An energy-efficient HTTP Adaptive Video Streaming with networking cost constraint over heterogeneous wireless networks. *IEEE Trans. on Multimedia*, 17(9):1646–1657, 2015.
[16] Anna Ukhanova, Evgeny Belyaev, Le Wang, and Søren Forchhammer. Power consumption analysis of constant bit rate video transmission over 3G networks. *Computer Communications*, 35(14):1695–1706, 2012.
[17] Mohammad Ashraful Hoque, Matti Siekkinen, and Jukka K Nurminen. Using crowd-sourced viewing statistics to save energy in wireless video streaming. In *MOBICOM*, pages 377–388, 2013.
[18] Yousef O Sharrab and Nabil J Sarhan. Aggregate power consumption modeling of live video streaming systems. In *MMSys*, pages 60–71, 2013.
[19] Kun Wang, Yun Shao, Lei Shu, and Guangjie Han. Ldpa: a local data processing architecture in ambient assisted living communications. *IEEE Communications Magazine*, 53(1):56–63, 2015.
[20] Ya Ju Yu, Pi Cheng Hsiu, and Ai Chun Pang. Energy-efficient video multicast in 4G wireless systems. *IEEE Trans. on Mobile Computing*, 11(10):1508–1522, 2012.
[21] Jang Ping Sheu, Chien Chi Kao, Shun Ren Yang, and Lee Fan Chang. A resource allocation scheme for scalable video multicast in WiMAX relay networks. *IEEE Trans. on Mobile Computing*, 12(1):90–104, 2013.

[22] Hui Wang, H Eduardo Roman, Liyong Yuan, Yongfeng Huang, and Rongli Wang. Connectivity, coverage and power consumption in large-scale wireless sensor networks. *Computer Networks*, 75:212–225, 2014.
[23] Nanhao Zhu and Athanasios V Vasilakos. A generic framework for energy evaluation on wireless sensor networks. *Wireless Networks*, 22(4):1199–1220, 2016.
[24] Peng Li, Song Guo, and Jiankun Hu. Energy-efficient cooperative communications for multimedia applications in multi-channel wireless networks. *IEEE Trans. on Computers*, 64(6):1670–1679, 2015.
[25] Kun Wang, Yihui Wang, Yanfei Sun, Song Guo, and Jinsong Wu. Green industrial Internet of Things architecture: An energy-efficient perspective. *IEEE Communications Magazine*, 54(12-Supp):48–54, 2016.
[26] Hauke Holtkamp, Gunther Auer, Samer Bazzi, and Harald Haas. Minimizing base station power consumption. *IEEE Journal on Selected Areas in Communications*, 32(2):297–306, 2014.
[27] Shuo Deng and Hari Balakrishnan. Traffic-aware techniques to reduce 3G/LTE wireless energy consumption. In *CoNEXT*, pages 181–192, 2012.
[28] Xiaoxia Zhang, Xuemin Sherman Shen, and Liang Liang Xie. Uplink achievable rate and power allocation in cooperative LTE-Advanced networks. *IEEE Trans. on Vehicular Technology*, 65(4):2196–2207, 2016.
[29] Junxian Huang, Feng Qian, Alexandre Gerber, Z Morley Mao, Subhabrata Sen, and Oliver Spatscheck. A close examination of performance and power characteristics of 4G LTE networks. In *MobiSys*, pages 225–238, 2012.
[30] Li-Ping Tung, Ying-Dar Lin, Yu-Hsien Kuo, Yuan-Cheng Lai, and Krishna M Sivalingam. Reducing power consumption in LTE data scheduling with the constraints of channel condition and QoS. *Computer Networks*, 75:149–159, 2014.
[31] Thomas Stockhammer. Dynamic adaptive streaming over HTTP –: standards and design principles. In *MMSys*, pages 133–144, 2011.
[32] Yan Zhang, N. Ansari, Mingquan Wu, and H. Yu. Afstart: An adaptive fast TCP slow start for wide area networks. In *ICC*, pages 1260–1264, 2012.
[33] In Kwan Yu and Richard Newman. TCP slow start with fair share of bandwidth. *Computer Networks*, 55(17):3932–3946, 2011.
[34] Monsoon power monitor. http://www.msoon.com/LabEquipment/PowerMonitor/.
[35] Dash industry forum. http://dashif.org/software.
[36] Network protocol analyzer — wireshark. https://www.wireshark.org.
[37] Qxdm state collection tool. https://www.qualcomm.com.
[38] Jetty web server. http://www.eclipse.org/jetty.
[39] Alex Zambelli. IIS smooth streaming technical overview. *Microsoft Corporation*, 3:40, 2009.