Energy-Aware CPU Frequency Scaling for Mobile Video Streaming

Wenjie Hu and Guohong Cao
Department of Computer Science and Engineering
The Pennsylvania State University
E-mail: {wwh5068, gcao}@cse.psu.edu

***Abstract*—The energy consumed by video streaming includes the energy consumed for data transmission and CPU processing, which are both affected by the CPU frequency. High CPU frequency can reduce the data transmission time but it consumes more CPU energy. Low CPU frequency reduces the CPU energy but increases the data transmission time and then increases the energy consumption. In this paper, we aim to reduce the total energy of mobile video streaming by adaptively adjusting the CPU frequency. Based on real measurement results, we model the effects of CPU frequency on TCP throughput and system power. Based on these models, we propose an Energy-aware CPU Frequency Scaling (EFS) algorithm which selects the CPU frequency that can achieve a balance between saving the data transmission energy and CPU energy. Since the downloading schedule of existing video streaming apps is not optimized in terms of energy, we also propose a method to determine when and how much data to download. Through trace-driven simulations and real measurement, we demonstrate that the EFS algorithm can reduce 30% of energy for the Youtube app, and the combination of our download method and EFS algorithm can save 50% of energy than the default Youtube app.**

I. Introduction

Video streaming has become extremely popular on mobile devices over the last few years. Mobile video streaming on Youtube, Netflix, has taken 55% of the total mobile data traffic in 2015, and will take 75% by 2020 [3]. Since video has much larger data size, a large amount of energy will be consumed to download video on smartphones. Thus, it is critical to improve the energy efficiency of video streaming on smartphones. The energy consumption of video streaming includes the energy consumed for data transmission and the energy consumed for CPU processing such as decoding. To reduce the data transmission energy, a widely used method is to download some amount of video content as fast as possible and then turn the wireless interface off [15, 6, 10]. Since the CPU energy is related to its working frequency [7, 13], it is possible to reduce the CPU energy by decreasing its frequency.

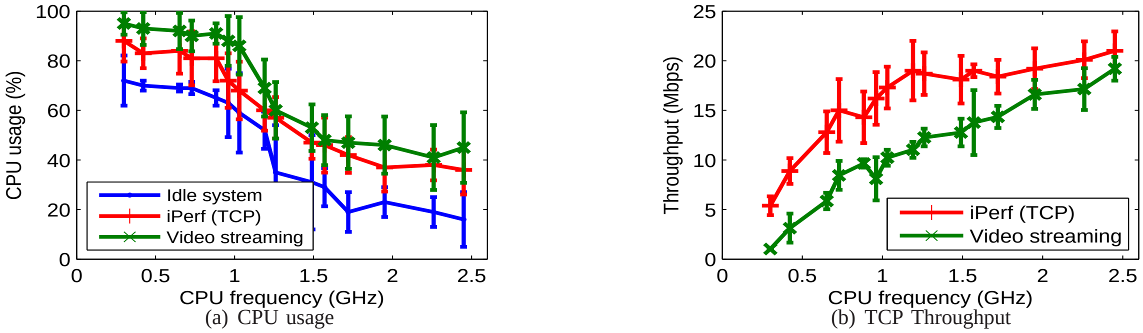
A straightforward method to save the energy consumption of video streaming is to reduce the data transmission energy and the CPU energy separately. However, these two goals are contradictory because the TCP throughput is related to the CPU frequency [20]. High CPU frequency can help increasing the TCP throughput and thus reducing the data transmission time, but costs much more CPU energy. On the other hand, low CPU frequency reduces the CPU energy, but makes the CPU a bottleneck and affects the TCP throughput. It increases the data transmission time and may increase the data transmission energy. To reduce the total energy of video streaming, the CPU frequency should be properly setup to achieve a balance between data transmission energy and CPU energy.

For modern smartphones, the CPU can work at a series of frequencies. The CPU frequency and the voltage provided to the CPU can be adjusted at run-time. This feature is called Dynamic Voltage and Frequency Scaling (DVFS). The system driver uses different policies to adjust the CPU frequency, which are called the *CPU governors*. For instance, the default CPU governor used by most smartphones is the *interactive* governor, which adjusts the CPU frequency according to the CPU usage. However, the default CPU governor tends to set the CPU at high frequency to provide better performance, which consumes a large amount of energy. Other CPU governors, such as the *powersave* governor, can restrict the CPU frequency to a low value, but they may increase the data transmission time and energy.

As of today there is no existing policy to reduce the total energy of mobile video streaming by properly adjusting the CPU frequency. In this paper, we aim to solve this problem. Based on real measurement results, we find that the CPU may become a bottleneck and affect the TCP throughput when its frequency is low, and then we model the effects of CPU frequency on TCP throughput and power consumption. Based on these models, we propose an Energy-aware CPU Frequency Scaling (EFS) algorithm which selects the CPU frequency that can achieve a balance between data transmission energy and CPU energy. Since the downloading schedule of existing video streaming apps is not optimized in terms of energy, we also propose a method to determine when and how much data to download. The efficiency of EFS algorithm and our downloading method is verified by trace-driven simulations and real measurement. Evaluation results show that the EFS algorithm can reduce 30% of energy, and the combination of our download method and EFS algorithm can save 50% of energy, when compared to the Youtube app.

The contribution of this paper can be summarized as follows.

- We are the first to study the relationship between TCP



(a) CPU usage

(b) TCP Throughput

Figure 1. The impact of CPU frequency on TCP throughput. When the CPU usage is higher than 70%, CPU becomes a bottleneck and affects TCP throughput.

throughput, system power and CPU frequency in video streaming. Based on measurement results, we model the effects of CPU frequency on TCP throughput and system power.

- We propose an Energy-aware CPU Frequency Scaling (EFS) algorithm to reduce the total energy for video streaming. During data transmission, EFS selects the most energy efficient CPU frequency considering both CPU energy and data transmission energy. When there is no data transmission, EFS selects a low CPU frequency to reduce the energy consumption without affecting the user experience.
- We consider the impact of the downloading schedule on energy and combine it with our EFS algorithm to further improve the energy efficiency of video streaming.

The rest of this paper is organized as follows. Section II introduces the background and the TCP throughput and power models. Section III presents the EFS algorithm and Section IV presents our energy efficient downloading schedule for video streaming. The evaluation results are shown in Section V. Section VI introduces the related work. Section VII concludes the paper.

II. Preliminaries

In this section, based on real measurement results, we model the effects of CPU frequency on TCP throughput and system power, and introduce our TCP throughput and power models.

A. Measurement Setup

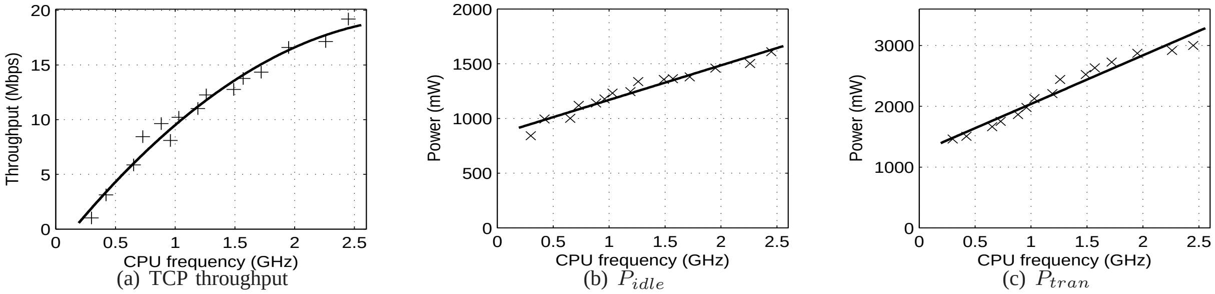
To model the impact of CPU frequency on TCP throughput and system power, we collect real measurement data related to TCP throughput and power consumption under different CPU frequencies. Our testbed is a rooted Samsung Galaxy S5, which is equipped with Qualcomm Snapdragon 801 CPU, which can work at 15 different frequencies from 300 MHz to 2.45 GHz. We use the 3C CPU Manager [1] to set the CPU working at a specific frequency and use the OS monitor to record the real time CPU usage.

The TCP throughput measurement is based on AT&T's LTE network. We use the Youtube app to watch a video with constant bit rate (720p) for 1 minute at different CPU frequencies. At each CPU frequency, we collect the network trace using TCPDUMP, which records the timestamp and data size of each packet. All packets with an interval less than 1 second are considered as one downloading period, and we compute the TCP throughput as the average value among all the downloading periods. Since the CPU can work at 15 frequencies, running a set of tests takes around 20 minutes. To measure the power consumption, we use the Monsoon power monitor to provide power directly to the smartphones, which can record the power value at a sample rate of 5000 Hz.

B. Impact of CPU on TCP Throughput

CPU usage is the percentage of CPU time used to process instructions, other than waiting. It is used to describe the load of the CPU. When the CPU usage is above 70%, it may become a bottleneck and affect the user experience. For smartphones, the operating system itself consumes a large amount of CPU. In Fig. 1(a), we show the CPU usage of idle system (all user applications are turned off). As can be seen, when the CPU frequency decreases, the CPU usage increases, which may affect the performance of user applications. Video streaming uses TCP as the transport layer protocol, and TCP uses lots of CPU capacity to handle congestion avoidance issues, buffer and reorder received packets, request the retransmission of missing packets, etc. On top of TCP, video streaming has complex application layer operations, such as moving data from the TCP buffer to the application buffer, decoding the received content and displaying them on screen, and thus requires more CPU capacity.

When the CPU frequency is low, the remaining CPU capacity may not be enough to process the TCP task and video streaming, and thus affecting the TCP throughput. To verify this, we measure the CPU usage and TCP throughput of two apps: iPerf (without application layer operation) and Youtube. As shown in Fig. 1, the CPU usage increases when



(a) TCP throughput (b) P_{idle} (c) P_{tran}

Figure 2. The TCP throughput and power model considering CPU frequency

its frequency decreases, and the TCP throughput decreases accordingly for both iPerf and Youtube. Also, the TCP throughput of Youtube is lower than that of iPerf and is more sensitive to the change of CPU frequency, as it has application layer operations. We also notice that the TCP throughput of iPerf increases quickly when the CPU frequency increases from 0.3GHz to 1.19GHz, but becomes almost flat after that, where the CPU is not a bottleneck. However, video streaming requires much more CPU capacity and hence the impact of CPU frequency on throughput is much higher.

C. TCP Throughput and Power Model

Here we mainly consider the TCP throughput of video streaming. The average value of TCP throughput at different CPU frequency is drawn in Fig. 2(a), and the TCP throughput $r(f)$ can be described as $r(f) = r_{max} \times r^*(f)$. The first part r_{max} is the network throughput which is not related to the variation of CPU frequency, and is only affected by the signal strength, location, the number of users nearby, etc. In this paper we model it using the average network throughput. More accurate measurement of network throughput can be found in [12, 22], which is out of the scope of this paper. The second part $r^*(f)$ describes the impact of CPU frequency on TCP throughput, which may vary with different phone models and the relationship can be modeled and trained offline.For example, the TCP throughput of our testbed is described as $r(f) = 19.19 \times (-0.12 \times f^2 + 0.71 \times f - 0.1)$.

In LTE, the wireless interface can work in four states: *idle*, *promotion*, *data transmission* and *tail*. Initially the LTE interface is in the idle state. When a data transmission request comes, it enters the promotion state to obtain the data transmission channels. Then it enter data transmission state to transmit data. After data transmission, the LTE interface is forced to stay in the tail state and wait for several seconds before going to the idle state. During the tail state, the phone still holds the data transmission channel, and can serve the next data transmission request immediately.

Since LTE has four states, we build four power models correspondingly. Our models describe the whole phone's power and use CPU frequency as an important parameter. Here we show an example of the idle power and data transmission power using different CPU frequency in Fig. 2(b) and Fig. 2(c) (the other two states show similar trend). As can be seen, the power in these states generally increases linearly with the CPU frequency. Note that in some previous work [13], the CPU power consumption increases super linearly with the frequency, since they only consider the power of CPU, where the voltage also changes linearly with the CPU frequency. Different from them, we consider the power consumption of the whole smartphones, where the voltage provided by the battery is a constant number. As a result, the power consumption changes linearly with the CPU frequency, similar to [7]. For our testbed, the power models of different states are summarized in Table I, where f is in GHz and power is in mW.

Table I
POWER MODEL CONSIDERING CPU FREQUENCY

State	Power (mW)	Duration (sec)
Idle	$P_{idle}(f) = 315.7f + 854$	-
Promotion	$P_{pro}(f) = 639.2f + 1206$	$t_{pro} = 0.91$
Data trans.	$P_{tran}(f) = 799.1f + 1241$	-
Tail	$P_{tail}(f) = 288.3f + 1119$	$t_{tail} = 10.35$

III. ENERGY-AWARE CPU FREQUENCY SCALING FOR EXISTING VIDEO STREAMING APPS

In this section, we introduce our Energy-aware CPU Frequency (EFS) algorithm to select the most energy efficient CPU frequency for existing video streaming apps.

A. Problem Statement

The video streaming process can be considered as a set of n data transmission tasks. Task T_i needs to download d_i data from time t_i. For an existing video streaming app, the downloading schedule is determined by the application, i.e., d_i and t_i can be considered as given value. To guarantee that the video is played smoothly, T_i must be downloaded before the next downloading period. In this paper we also call t_{i+1} as the task end of T_i. For the last task T_n, the task end is the time when the whole video is played out. The duration from the start to the end of a task is called the length of a task. The energy consumption of task T_i is defined as the total energy consumed from the start to the

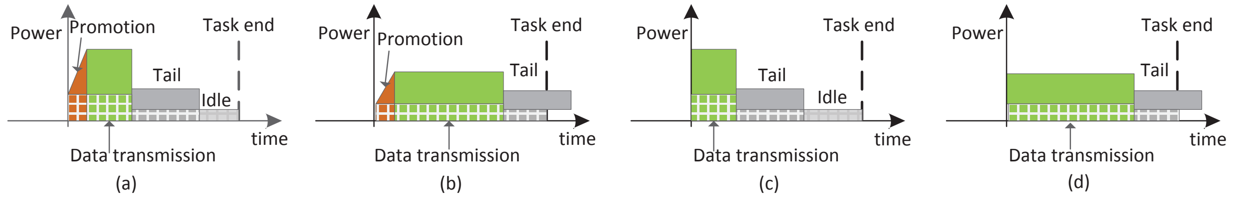


Figure 3. The four cases to compute the energy of a task, which is defined as the total energy from the start time to the task end. The dotted filling indicates the CPU power in the corresponding state.

end, and our goal is to minimize the total energy of all the tasks, which can be described as **minimize** $\sum_{i=1}^{n} E(T_i)$.

B. The Energy Consumption of One Task

The energy consumption of a task contains the data transmission energy and the CPU energy. Based on the starting and finishing states of a task, the energy consumption of T_i can be computed in four cases, as illustrated in Fig. 3. In each case, we assume the CPU works at one frequency during data transmission and a lower frequency when there is no data transmission. During data transmission, we select a frequency from the CPU frequency set F to achieve a good tradeoff between reducing the data transmission energy and the CPU energy. When the data transmission is done, the CPU works at a lower frequency f_{min} which reduces the CPU energy and also provides satisfactory performance. We do not consider the case where the data transmission cannot be finished before the task ends, since it violates the downloading schedule and then affects user experience [17].

Case (a): As shown in Fig. 3 (a), the LTE interface is in the idle state at the beginning of T_i. Thus, it enters promotion state first and pay extra promotion energy. In this case, the CPU works at relative high frequency (f_a) and the TCP throughput is relative high ($r(f_a) > \frac{d_i}{l_i - t_{tail} - t_{pro}}$), so the phone demotes to idle state at the end of the task. The total energy can be computed using Eq. 1.

$$\begin{aligned} E(T_i)^a = & t_{pro} \times P_{pro}(f_a) + \frac{d_i}{r(f_a)} \times P_{tran}(f_a) \\ & + t_{tail} \times P_{tail}(f_{min}) \\ & + (l_i - t_{pro} - t_{tail} - \frac{d_i}{r(f_a)}) \times P_{idle}(f_{min}), \\ & \text{if } r(f_a) > \frac{d_i}{l_i - t_{tail} - t_{pro}} \end{aligned} \tag{1}$$

Case (b): The energy consumption of Case (b) is shown in Fig. 3 (b). Similar to Case (a), the wireless interface is in the idle state at the beginning of T_i. However, the TCP throughput in this case is low as low CPU frequency is selected. At the end of task T_i, the LTE interface is still in the tail state. The total energy is computed as Eq. 2. In this case the next task can skip the promotion state and start data transmission immediately.

$$\begin{aligned} E(T_i)^b = & t_{pro} \times P_{pro}(f_b) + \frac{d_i}{r(f_b)} \times P_{tran}(f_b) \\ & + (l_i - t_{pro} - \frac{d_i}{r(f_b)}) \times P_{tail}(f_{min}), \\ & \text{if } \frac{d_i}{l_i - t_{pro}} < r(f_b) \le \frac{d_i}{l_i - t_{tail} - t_{pro}} \end{aligned} \tag{2}$$

Case (c): As shown in Fig. 3 (c), the wireless interface is in high power state at the beginning, so data transfer starts immediately. Similar to Case (a), the TCP throughput is assumed to be high in this case ($r(f_c) > \frac{d_i}{l_i - t_{tail}}$). Then the total energy is computed as Eq. 3.

$$\begin{aligned} E(T_i)^c = & \frac{d_i}{r(f_c)} \times P_{tran}(f_c) + t_{tail} \times P_{tail}(f_{min}) \\ & + (l_i - t_{tail} - \frac{d_i}{r(f_c)}) \times P_{idle}(f_{min}), \\ & \text{if } r(f_c) > \frac{d_i}{l_i - t_{tail}} \end{aligned} \tag{3}$$

Case (d): The energy consumption of Case (d) is shown in Fig. 3 (d), where the data transmission starts immediately, similar to Case (c). But the TCP throughput is assumed to be low and the LTE interface is still in the tail state at the task end. The total energy is computed as Eq. 4.

$$\begin{aligned} E(T_i)^d = & \frac{d_i}{r(f_d)} \times P_{tran}(f_d) \\ & + (l_i - \frac{d_i}{r(f_d)}) \times P_{tail}(f_{min}), \\ & \text{if } \frac{d_i}{l_i} < r(f_d) \le \frac{d_i}{l_i - t_{tail}} \end{aligned} \tag{4}$$

For task T_i, we compute the energy in all of the four cases. In each case we search for the CPU frequency that can minimize the energy. Then we define the minimum energy in the four cases as the min energy of task T_i, as shown in Eq. 5.

$$\begin{aligned} & E(T_i) \in \{\min E(T_i)^a, \min E(T_i)^b, \min E(T_i)^c, \min E(T_i)^d\} \\ & f_a, f_b, f_c, f_d \in F \end{aligned} \tag{5}$$

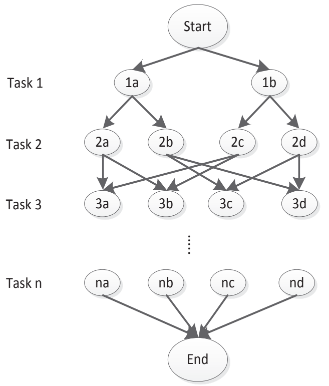


Figure 4. Mapping the minimum energy of video streaming to the shortest path problem

C. Energy-Aware CPU Frequency Scaling Algorithm

For each task, it is easy to obtain the minimum energy as there are only four cases. However, since the ending of previous tasks also affects the energy of later tasks, minimizing the energy of every task individually may not minimize the total energy of all tasks. To solve this problem, we propose an energy-aware CPU frequency scaling (EFS) algorithm which aims to find the global optimal solution. Our key idea is to map this problem to the shortest path problem.

We build a directed graph as shown in Fig. 4. For each task, there are four cases to compute the energy, as illustrated in Fig. 3, except the first one, which only has two cases since the wireless interface is in the idle state at the beginning. Each energy case of a task is mapped to one node in the graph. For example, the two cases of task 1 map to nodes $1a$ and $1b$. Besides these nodes, we add a virtual start and virtual end node. Next we add links to the graph. Assuming task T_i is downloaded using Case (a), then the LTE interface enters idle state at the end, so task T_{i+1} can only be scheduled by Case (a) or Case (b). Thus, we add directed links from node ia to nodes $(i+1)a$ and $(i+1)b$. The other links between task nodes are added similarly. For the two virtual nodes, we add links from the virtual start node to the two cases of task 1, and add links from the four cases of task n to the virtual end node. The weight of a link is the energy consumption of the node at the end of the link. For the four links from task n to the virtual end node, their weight is defined as 0. In this graph, we take into account all power cases of each task and all possible schedule paths between tasks, so each path from the virtual start node to the virtual end node will map to one schedule of all tasks, and vice versa. As a result, the minimum energy of all tasks corresponds to the shortest path from the virtual start node to the virtual end node.

Based on the graph we can use the Dijkstra algorithm to find out the shortest path. Given n tasks, the number of nodes in the graph is $O(4n)$, and the number of edges is $O(8n)$. As the time complexity of the shortest path algorithm is $O(V+E)logV$, where V is the number of nodes and E is the number of edges, then the time complexity is $O(n\log n)$ in our case. Additionally, as the CPU can work at $|F|$ discrete frequencies, computing the weight of each link need to consider all the $|F|$ possibilities. Putting them together, the time complexity of the EFS algorithm is $O(|F|n\log n)$.

Table II
THE MINIMUM CPU FREQUENCY FOR VIDEO STREAMING WHEN THE WIRELESS INTERFACE IS TURNED OFF

Video resolution	Min CPU frequency
360p	422 MHz
480p	652 MHz
720p	652 MHz
1080p	883 MHz

D. Minimum CPU Frequency Selection

In previous sections, we assume the minimum CPU frequency without data transmission (f_{min}) is a constant value. In fact, this value is related to the video resolution. To obtain this minimum CPU frequency under a specific video resolution, one simple solution is to play videos from the local storage, and then measure the CPU frequency. However, this minimum CPU frequency would be smaller than what is needed. During video streaming, the system also needs to maintain the buffer and TCP connection even when there is no data transmission. There may be some background apps that will consume extra CPU capacity, so the system will require a higher CPU frequency.

To solve this problem, we use Youtube to stream a video at the given resolution and pause it to buffer a long period of video. Then, we tune the CPU frequency and search for the minimum frequency that can still play the buffered content smoothly. The results for different video resolutions are shown in Table II. Note that during our measurement, the background apps are still running as normal, and their CPU capacity requirement has also been considered.

IV. ENERGY-AWARE DOWNLOADING SCHEDULE FOR VIDEO STREAMING

The downloading schedule of video streaming determines when and how much data to download. However, the downloading schedule of existing apps is not optimized to reduce energy. In this section, we design an energy efficient downloading schedule and combine it with our energy-aware CPU frequency scaling algorithm.

A. How Much to Download

Given a video size D, we can estimate its playback time based on the bitrate of the video. The bitrate v is related to the video resolution. For example, the 480p video has a bitrate of 1 Mbps and the 720p video has a bitrate of 2.5 Mbps [2]. The playback time for the video content is around D/v. The energy consumed by downloading D size of data may have four cases, as shown in Fig. 3. The minimum

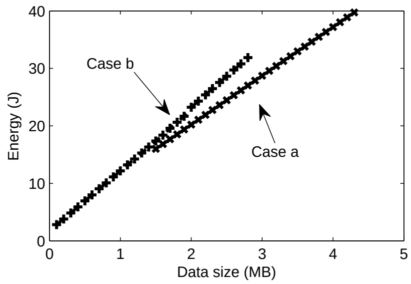


Figure 5. The minimum energy to download video content with different data size

energy of using Case (a) and Case (b) to download different size of data is shown in Fig. 5. For both cases, the energy consumption is a straight line increasing with the data size. We call the data size when $E(T)^a = E(T)^b$ as β, and it is 1.5 MB in our case. As the energy of Case (c) and Case (d) is smaller than that of Case (a) and Case (b) by a constant value (the promotion energy), they are not considered here.

From Fig. 5, we can see that when the downloading data size is less than a threshold β, it should be downloaded by Case (b), i.e., it should be downloaded with a smaller throughput for a longer time. When the data size is larger than β, then it can be downloaded directly or divided into multiple pieces with each piece smaller than β. However, the energy per byte in Case (a) (the slope of the line) is much smaller than that of Case (b), and thus using Case (a) to download the same size of data is more energy efficient. Therefore, when the video content is larger than β, it should be downloaded in one piece using Case (a). This conclusion is also consistent with previous works [10, 6]. Considering the buffer to hold the video content on smartphones is limited, we set the optimal downloading data size to the maximum buffer size.

B. When to Download

As described in previous section, when the downloading data size D is larger than β, it is more energy efficient to download using Case (a), and then the LTE interface enters the idle state before the task ends. To save energy, the LTE interface should stay in the idle state as long as possible. On the other hand, the video content should be downloaded before being used to provide better quality of experience (QoE). Thus, the next downloading should start a little earlier. The smallest decoding unit in video is called Group of Pictures (GoP), which has a fixed length according to the video coding protocol and frame organization [10]. Suppose this length is g, then the data size within the length of a GoP is approximately $v \times g$, where v is the bitrate of the video. As the CPU frequency used in the next downloading period is not known beforehand, we should consider the worst case, where the CPU frequency is f_{min} and the TCP throughput is $r(f_{min})$. To guarantee one GoP of the next task is downloaded before the end of the current task, the interval between tasks is computed as Eq. 6.

Table III
VIDEO BENCHMARK

Video id	Length (sec)	Data size (MB)	Resolution
1	57	8.8	720p
2	163	20.3	480p
3	271	53.3	720p
4	301	39.7	480p
5	496	79.9	720p
6	594	56.1	480p

$$Interval = \frac{D}{v} - t_{pro} - \frac{v \times g}{r(f_{min})} \qquad (6)$$

V. EVALUATIONS

In this section we use trace-driven simulations to demonstrate that our energy-aware CPU frequency scaling algorithm can help existing video streaming apps to save energy, and more energy can be saved using the optimized downloading schedule.

A. Simulation Setup

The trace used for simulation is collected from the Youtube app running on Samsung Galaxy S5. We watch a group of videos with different length, data size and resolution, as listed in Table III. We mainly consider videos less than 10 minutes since videos longer than 10 minutes are rare [4]. We collect two kinds of traces: the network trace, which is used to extract the downloading time and downloading data size, and the real-time CPU frequency trace, which is read from the file "`scaling_cur_freq`". Based on these traces, we mainly compare the performance of the following methods.

- `Youtube`: the original Youtube app using the default interactive CPU governor to adjust the CPU frequency.
- `Youtube+MaxMin`: the Youtube app uses the highest CPU frequency during data transmission and the minimum CPU frequency without data transmission.
- `Youtube+EFS`: the Youtube app using our *Energy-aware Frequency Scaling algorithm (EFS)* to adjust CPU frequency.
- `Ourstreaming+EFS`: the combination of optimized downloading schedule and the EFS algorithm. The buffer size is set to 10 MB.

B. Energy Comparison

We first compare the whole phone's energy consumption of different methods when watching videos in Table III, and show the results in Fig. 6. As can be seen, the energy consumption generally increases when the video length increases. This is because we consider the energy consumption during the whole playback period of the video. The MaxMin method saves a large amount of energy and the EFS method saves more. The combination of ourstreaming

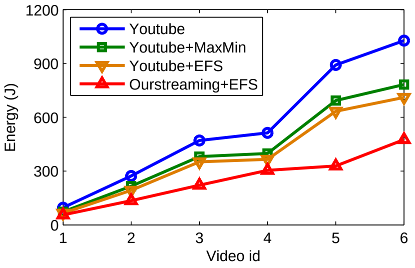


Figure 6. The total energy consumption of different methods

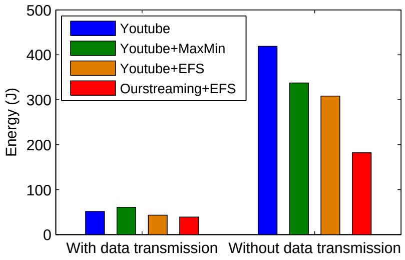


Figure 7. Impact of CPU frequency

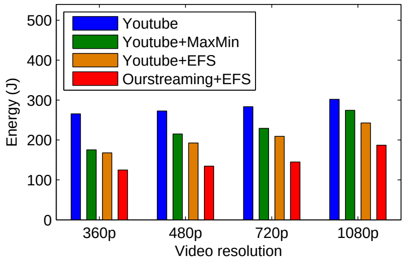


Figure 8. Impact of video resolutions

and EFS can save much more energy than simply using the EFS algorithm. When the video is longer and the data size is larger, more energy can be saved, because there are more downloading tasks and thus more opportunities to adjust the CPU frequency. On average, the MaxMin method and the EFS method can save 22.1% and 30.2% more energy than the default Youtube method, respectively. The combination of ourstreaming and EFS can save 50.6% of energy. On top of EFS, our optimized downloading schedule helps to save another 29.2% energy. We also notice that the energy consumption of MaxMin and EFS has similar trends as that of the default Youtube method, since they use the same downloading schedule. Ourstreaming adjusts the downloading schedule by transmitting multiple tasks together and thus shows a different trend. The energy of ourstreaming method is mainly related to the data size and the length of a video.

C. Impact of CPU Frequency on Energy

To better understand the energy saved by selecting different CPU frequency, we divide the total time into two time periods: the period with data transmission (promotion and data transmission time) and the period without data transmission (tail and idle time), and then analyze the impact of CPU frequency on energy in these two time periods. The comparison of these two parts of energy of video 3 is shown in Fig. 7. As can be seen, the EFS algorithm can help save the data transmission energy, because it selects a proper CPU frequency. Specifically, Youtube+EFS saves 13.1% of energy and oustreaming+EFS saves 24.2% of energy during data transmission. However, MaxMin consumes a little more energy than the Youtube method since it always selects the highest (most power consuming) CPU frequency. When there is no data transmission, both MaxMin and EFS can save energy when compared to the Youtube method, because they all use lower CPU frequency. The EFS method reduces more energy than MaxMin because it spends less time in the period when there is no data transmission.

D. Impact of Video Resolution

Since mobile devices have different screen resolutions and different network speed, video providers generally provide multiple versions for the same video with different resolutions to satisfy users' requirements. The video client can select a fixed resolution or use DASH technology to dynamically adjust the video resolution. To test the performance of different methods under different video resolutions, we collect traces of video 2 with a resolution of 360p, 480p, 720p and 1080p, respectively. The energy consumption of different methods is shown in Fig. 8. Clearly, MaxMin and EFS can save energy under all resolutions, and ourstreaming+EFS can save more. Considering the energy saving ratio, we can see that it decreases when the video resolution increases. When watching the 360p version of video, MaxMin, EFS and ourstreaming+EFS can save 33%, 36.8% and 52.9% of energy when compared to the Youtube method, respectively. However, these saving ratios drop to 9%, 19.7% and 38.1% when watching the 1080p video. The reason is that a video with higher resolution has larger data size and more pixels, and thus all methods need to select a higher CPU frequency to download the video on time, decode and play smoothly. As a result, the difference between their CPU frequencies and the default system is smaller and less energy can be saved.

VI. Related Work

The wireless interface, especially the cellular interface consumes a lot of power on smartphones [16]. In cellular networks, the wireless interface stays in a high power state (tail state) after a data transmission, and the tail state wastes a large amount of energy. To save energy, researchers propose to aggregate data tasks together to amortize the tail energy [23, 9, 8]. Similar idea is also used by video streaming which downloads a group of video content together and then turn the wireless interface off [15]. However, video streaming has strict delay constraint and the data should be downloaded before being used [10, 11]. EVIS uses multiple networks to provide energy-efficient and quality-guaranteed video streaming [21], but it does not consider the impact of CPU frequency to the network throughput.

Video streaming requires lots of CPU processing power to provide good QoE. The CPU energy is related to its working frequency [7, 19]. High CPU frequency can provide better performance but consumes more energy. Many solutions have been proposed to adjust the CPU frequency to achieve a balance between performance and energy [18, 5, 24, 14]. They have some interesting results, such as how to select the CPU frequency to finish the tasks before their deadline and save energy [14], however, none of them considers the impact of CPU frequency on TCP throughput.

The energy consumed by video streaming includes data transmission energy and CPU energy. This makes the problem more complex, since the TCP throughput is closely related to CPU frequency [20]. High CPU frequency increases the CPU energy consumption, while low CPU frequency increases the data transmission time and may increase the data transmission energy. Kwak *et al.* consider the tradeoff between saving CPU energy and data transmission energy in [13], and suggest to reduce the CPU frequency when the network becomes bottleneck. Their intuition is to save the CPU energy when waiting for the network tasks. However, they do not consider that the TCP throughput is also reduced when reducing the CPU frequency. Thus, their solution may introduce too much delay for video streaming applications. In this paper, we consider the delay and set the CPU to a proper frequency that can save energy and ensure the video content is downloaded before being used.

VII. Conclusion and Future Work

In this paper, we modeled the effects of CPU frequency on TCP throughput and system power, and studied how to save energy for video streaming considering the CPU frequency. During video streaming, high CPU frequency can reduce the data transmission time but it consumes more CPU energy; low CPU frequency reduces the CPU energy but increases the data transmission time and then increase the energy consumption. To address this problem, we proposed an energy-aware CPU frequency scaling algorithm (EFS) which can properly adjust the CPU frequency to reduce the overall energy during video streaming. This algorithm can be directly applied to existing video streaming apps, like Youtube. Also, the downloading schedule of existing apps is not optimized in terms of energy. We address this problem by proposing an energy efficient downloading schedule, which can save more energy when combined with the EFS algorithm. Based on trace-driven simulations and real measurement, we demonstrate that EFS can save 30% of energy than the default Youtube app. By using properly selected downloading data size and downloading interval in our EFS algorithm, more than 50% of energy can be saved when compared to the default Youtube app.

Acknowledgment

This work was supported in part by the National Science Foundation (NSF) under grants CNS-1526425 and CNS-1421578.

References

[1] 3c cpu manager. https://goo.gl/2OoLMF.

[2] Video bitrate. http://www.lighterra.com/papers/videoencodingh264/.

[3] Cisco visual networking index: Global mobile data traffic forecast update, 2015-2020. http://goo.gl/DXWFyr, 2015.

[4] X. Cheng, C. Dale, and J. Liu. Statistics and Social Network of YouTube Videos. In *16th International Workshop on Quality of Service (IWQoS)*, 2008.

[5] K. Choi, R. Soma, and M. Pedram. Fine-grained Dynamic Voltage and Frequency Scaling for Precise Energy and Performance Tradeoff based on the Ratio of Off-chip Access to On-chip Computation Times. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(1):18–28, 2005.

[6] M. A. Hoque, M. Siekkinen, and J. K. Nurminen. Using Crowd-sourced Viewing Statistics to Save Energy in Wireless Video Streaming. In *ACM MobiCom*, 2013.

[7] C.-H. Hsu, U. Kremer, and M. Hsiao. Compiler-directed Dynamic Voltage/Frequency Scheduling for Energy Reduction in Microprocessors. In *Proceedings of the 2001 International Symposium on Low Power Electronics and Design (ISLPED)*, 2001.

[8] W. Hu and G. Cao. Energy Optimization Through Traffic Aggregation in Wireless Networks. In *IEEE INFOCOM*, 2014.

[9] W. Hu and G. Cao. Quality-Aware Traffic Offloading in Wireless Networks. In *ACM MobiHoc*, 2014.

[10] W. Hu and G. Cao. Energy-Aware Video Streaming on Smartphones. In *IEEE INFOCOM*, 2015.

[11] Krishnan, S. Shunmuga and Sitaraman, Ramesh K. Video stream quality impacts viewer behavior: Inferring causality using quasi-experimental designs. In *ACM IMC*, 2012.

[12] S. Kumar, E. Hamed, D. Katabi, and L. Erran Li. LTE Radio Analytics Made Easy and Accessible. In *ACM SIGCOMM*, pages 211–222, 2014.

[13] J. Kwak, O. Choi, S. Chong, and P. Mohapatra. Dynamic Speed Scaling for Energy Minimization in Delay-Tolerant Smartphone Applications. In *IEEE INFOCOM*, 2014.

[14] K. Kwon, S. Chae, and K.-G. Woo. An application-level energy-efficient scheduling for dynamic voltage and frequency scaling. In *2013 IEEE International Conference on Consumer Electronics (ICCE)*, 2013.

[15] X. Li, M. Dong, Z. Ma, and F. C. Fernandes. GreenTube: Power Optimization for Mobile Videostreaming via Dynamic Cache Management. In *Proceedings of the 20th ACM International Conference on Multimedia*, 2012.

[16] R. Mittal, A. Kansal, and R. Chandra. Empowering Developers to Estimate App Energy Consumption. In *ACM MobiCom*, 2012.

[17] H. Nam, K. H. Kim, and H. Schulzrinne. QoE Matters More Than QoS: Why People Stop Watching Cat Videoss. In *IEEE INFOCOM*, 2016.

[18] P. Pillai and K. G. Shin. Real-time Dynamic Voltage Scaling for Low-power Embedded Operating Systems. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles (SOSP)*, pages 89–102, 2001.

[19] X. Ruan, X. Qin, Z. Zong, K. Bellam, and M. Nijim. An Energy-Efficient Scheduling Algorithm Using Dynamic Voltage Scaling for Parallel Applications on Clusters. In *IEEE ICCCN*, 2007.

[20] S. Sanadhya and R. Sivakumar. Rethinking TCP Flow Control for Smartphones and Tablets. *Wireless Networks*, 20(7):2063–2080, Oct. 2014.

[21] J. Wu, B. Cheng, and M. Wang. Energy Minimization for Quality-Constrained Video with Multipath TCP over Heterogeneous Wireless Networks. In *IEEE ICDCS*, 2016.

[22] X. Xie, X. Zhang, S. Kumar, and L. E. Li. piStream: Physical Layer Informed Adaptive Video Streaming over LTE. In *ACM MobiCom*, pages 413–425, 2015.

[23] B. Zhao, W. Hu, Q. Zheng, and G. Cao. Energy-Aware Web Browsing on Smartphones. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 26(3):761–774, 2015.

[24] J. Zhuo and C. Chakrabarti. Energy-efficient Dynamic Task Scheduling Algorithms for DVS Systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 7(2):17:1–17:25, 2008.