GreenTube: Power Optimization for Mobile Video Streaming via Dynamic Cache Management

Xin Li
Dept. of Computer Science
George Mason University
Fairfax, VA
xlih@gmu.edu

Mian Dong, Zhan Ma, Felix Fernandes
Dallas Technology Lab
Samsung Telecommunications America
Richardson, TX
{mian.dong, zhan.ma, felix.f}@samsung.com

ABSTRACT

Mobile video streaming has become one of the most popular applications in the trend of smartphone booming and the prevalence of 3G/4G networks, i.e., HSPA, HSPA+, and LTE. However, the prohibitively high power consumption by 3G/4G radios in smartphones reduces battery life significantly and thus severely hurts user experience. To tackle this challenge, we designed *GreenTube*, a system that optimizes power consumption for mobile video streaming by judiciously scheduling downloading activities to minimize unnecessary active periods of 3G/4G radio. GreenTube achieves this by dynamically managing the downloading cache based on user viewing history and network condition. We implemented GreenTube on Android-based smartphones. Experimental results show that GreenTube achieves large power reductions of more than 70% (on the 3G/4G radio) and 40% (for the whole system). We believe GreenTube is a desirable upgrade to the Android system, especially in the light of increasing LTE popularity.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Measurement techniques, Design studies, Performance attributes; C.2.1 [**Network Architecture and Design**]: wireless communication

Keywords

Power Optimization, Mobile Video Streaming

1. INTRODUCTION

Power optimization for smartphones continues to be a relevant problem. The problem is becoming pronounced, especially with the rapid growth of mobile data usage for a large variety of multimedia applications, including games, music, videos, etc. The 3G/4G networks, such as HSPA, HSPA+, and LTE, offer their users much faster mobile data access speeds and enable data-intensive applications such as HD video streaming. However, the wireless communications also impose a significant power cost on smartphones [1].

Our goal is *to make mobile video streaming more power efficient over 3G/4G networks*. Not only is video streaming among the most popular smartphone apps (applications) according to recent studies [2] [19], but also its data-intensive nature indeed leads to extremely high power consumption. For example, the average power consumption by Galaxy Nexus is 2.3 W to play a 720p video from HTTP streaming via Verizon LTE network according to our measurement. As a result, a battery of full capacity (1850 mAh) can only support three hours of video streaming. Such prohibitively high power consumption severely hurts user experience.

We choose 3G/4G radio as our major target for power optimization because 3G/4G radio contributes a significant portion of system power consumption. In comparison to the previous example, the average power consumption by Galaxy Nexus is only 1.1 W to play the same video from SD card. In other words, more than 50% of the system power is consumed by the LTE radio. Thus, optimizing power consumption of 3G/4G radio is essential for maximizing the smartphone battery life and improving user experience. To the best of our knowledge, this paper is the first to discuss power optimization for mobile video streaming via 3G/4G networks. As the first study, we must answer the following research questions.

First, why does the 3G/4G radio consume so much power in video streaming? Through control experiments and video streaming trace analysis, we show that a significant amount of power is consumed due to the conflict between the video streaming support in state-of-the-art smartphone systems and the power control mechanism in 3G/4G networks. That is, the time interval between successive downloading sessions is too short for the radio to enter power saving mode, or idle state, defined in 3G/4G networks. As a result, the 3G/4G radio has to remain active throughout the video streaming session even if there is no downloading session going on.

Second, will the problem be solved by using a large downloading cache? A natural solution to solve the above problem is to download the whole video into a large cache and turn off the radio when the downloading session ends. This approach will work if the user chooses to watch through the whole video. However, according to our study, 80% of YouTube sessions in smartphones are less than half of the corresponding video durations. As a result, downloading the whole video will lead to waste in both data usage and power.

Finally, will the problem be solved by traffic reshaping?

MM'12, October 29–November 2, 2012, Nara, Japan.

Area Chair: Liangliang Cao

Many WiFi power optimization works [11] [4] propose to reshape the wireless traffic and create sufficient long periods for WiFi radios to enter power saving mode. Directly using this approach, however, does not work for 3G/4G networks because the 3G/4G radios, unlike their WiFi counterparts, simply can not enter power saving mode immediately after traffic ends as defined in 3G/4G standards. Instead, they need to wait for a fairly long period, e.g., 10 seconds in Verizon LTE network, before entering power saving mode. The length of such period is comparable to that of each downloading session. Therefore, it requires judicious scheduling of each downloading session to effectively reduce power consumption of 3G/4G radios.

In this paper, we present *GreenTube*, a system that optimizes power consumption for mobile video streaming by judiciously scheduling downloading activities to minimize unnecessary active period of 3G/4G radios. GreenTube achieves this via dynamically managing the downloading cache based on user viewing history and network condition. We implemented GreenTube on Android-based smartphones and experimental results show large power reductions of more than 70% (on the 3G/4G radio) and 40% (for the whole system).

In designing and realizing GreenTube, we make the following contributions:

- **Characterize the problem of conflict between the power state transition defined in 3G/4G networks and the video streaming support in state-of-the-art smartphone systems.** Through three motivational studies we show the significant energy impact of such a conflict and summarize the design requirements of a system that can address this issue (Section 3).
- **Design a light-weight system, GreenTube, to reduce power consumption.** The design of GreenTube extensively leverages the findings from the motivational studies in order to meet the design requirements (Section 4).
- **Implement and evaluate GreenTube on Android-based smartphones.** Promising power improvements provide confidence that GreenTube is an important step towards energy-efficient mobile video streaming via 3G/4G networks (Section 5).

The rest of this paper is organized as follows. Section 2 provides necessary background information. Section 3 motivates the GreenTube design through three studies, followed by the system design in Section 4. Section 5 presents system implementation and evaluation. Section 6 surveys related work. Section 7 discusses the limitations and future work. Section 8 concludes this paper.

2. BACKGROUND

We start with a brief overview of mobile video streaming, then introduce power states defined in 3G/4G network, and finally discuss the video streaming support in state-of-the-art smartphone systems.

2.1 Mobile Video Streaming Overview

A video streaming system can be implemented in many ways. It can be built using a conventional Content Delivery Network (CDN) architecture such as YouTube, Hulu, etc,

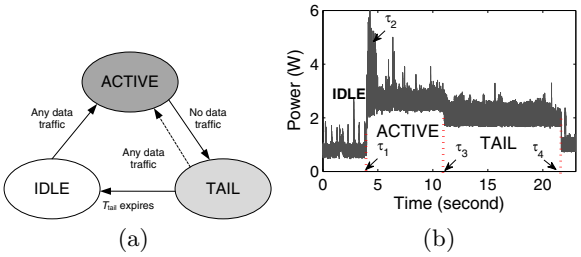

Figure 1: 3G/4G RRC and power states, (a) simplified RRC state transition; (b) power trace

or purely peer-to-peer (P2P) such as PPLive, PPStream, or even hybrid CDN-P2P [21]. HTTP, RTSP, P2P protocols are implemented to realize the successful streaming system. Among them, our focus is the popular and dominant HTTP video streaming. For instance, YouTube, Hulu, etc, use the HTTP streaming protocol for content delivery. One promising feature of the HTTP protocol is the support for resuming the broken or interrupted downloads. This enables our system design to close TCP connection and thus improve power consumption.

Video delivered over the Internet can be encapsulated using many container formats such as AVI, FLV, MP4, MKV, etc. FLV was dominant in the past. Recently, however, streaming video contents are increasingly contained in MP4 format, rather than FLV, especially for HD contents. To support our study, we used a crawler to download 3 Terabytes of the most popular YouTube videos over a 45-day span. From this massive dataset, videos with 720p and 1080p resolutions are all encapsulated in the MP4 container. Like other modern container formats, MP4 allows streaming over the Internet. We use MP4 videos exclusively in this paper.

2.2 Power States in 3G/4G Network

We next cover the necessary background on state machine behavior and corresponding power characteristics of the 3G/4G network.

Both 3G and 4G networks share the similar generalized radio resource control (RRC) states, IDLE, ACTIVE and TAIL as shown in Figure 1(a). For a typical data exchange, assuming the device starts at IDLE state, current state is promoted to the ACTIVE state for data transmission with a certain delay (i.e., noted as promotion delay [9, 18]). After data transmission, it is usually demoted from the ACTIVE state requiring higher radio resource and radio power consumption to the TAIL state with lower radio resource and power consumption, until the tail timer (noted as T_{tail}) expires and finally puts the device into the IDLE state.

Given the state machine for 3G/4G networks, the power trace on smartphones can be easily explained. As shown in Figure 1(b) for video streaming over Verizon LTE network, we observe that the network activities accurately match the different states with associated power levels. Smartphones consume almost zero power in IDLE state. Power consumption rises when the smartphone is promoted from the IDLE state to ACTIVE after sending a packet at time τ_1. After a certain promotion delay, the device starts active data transfer at time τ_2, until it enters the TAIL state with lower power at τ_3. After the tail timer expires at τ_4, the smart-

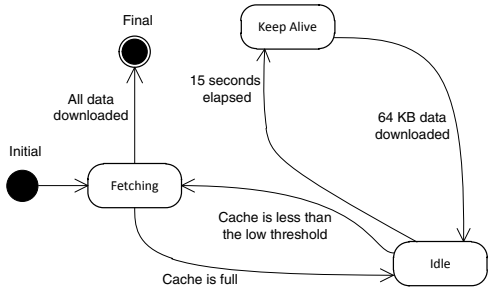

Figure 2: Android HTTP data source provider state machine

phone enters IDLE and the allocated radio resource is released. Note that our findings are consistent with the work presented in [9].

2.3 HTTP Streaming Support in Smartphones

We next introduce HTTP streaming support in Android, one of the most popular smartphone OSes.

In the Android Multimedia Framework, the major components involved in video playback include data source providers (like disk file, HTTP streaming, etc), video container demuxers, audio/video codecs and a media player. The HTTP streaming downloading behaviour is largely determined by the data source provider which interacts with the HTTP server and feeds the data to the media player. The HTTP streaming data source provider is stateful and there are three states: *Fetching*, *Idle* and *Keep Alive*. In the data source provider, the video data downloaded from the HTTP server is held in a large memory cache, or downloading cache. The size of the cache is bounded by a high threshold. There is also a low threshold, the lower bound of the cache, which is used to improve the playback smoothness. By default, the high threshold and low threshold are is set to 20 MB 4 MB respectively.

During an HTTP video streaming session, the data is downloaded periodically. We now explain this process in detail using the state machine in Figure 2. Initially, when the user starts a video streaming session, the data source provider enters into the *Fetching* state and continuously downloads the video data from the HTTP server. When the cache size reaches the high threshold, it transits to the *Idle* state. By default, while in the *Idle* state, a keep-alive mechanism is automatically triggered every 15 seconds to transition into the *Keep Alive* state. On entering into the *Keep Alive* state, 64 KB video data is downloaded and then the state transits back into the *Idle* state. Once the cached video content size falls below the designated low threshold, the state moves to *Fetching* and another continuous downloading begins.

The different states of the HTTP data source provider can be mapped to the different states of the 3G/4G networks. In other words, the different states correspond to different power consumption states. As a result, the power consumption during video playback is largely determined by this periodic downloading behaviour. This relationship is further studied and explored in the following sections.

One exception worth mentioning is that although the YouTube app directly utilizes the Android Multimedia Framework for video playback, it uses an undocumented feature which disables the keep-alive mechanism and closes the TCP connection every time the cache is filled up. This effectively eliminates all data traffic between the HTTP server and the smartphone in the *Idle* state.

Table 1: Smartphone Specifications

Model	Galaxy S	Galaxy S II	Galaxy Nexus
Processor	Hummingbird	Snapdragon	OMAP4
Memory	512MB	1GB	1GB
Network	ATT HSPA	Tmobile HSPA+	Verizon LTE
Android	2.3.6	2.3.6	4.0.3

3. MOTIVATION

We next report three studies that directly motivate the design of GreenTube: mobile video streaming power characterization, YouTube usage by smartphone users, and LTE network speed measurement.

3.1 Power Characterization

3.1.1 Methodology

Video Content: As briefly mentioned earlier, we developed a YouTube crawler that automatically downloads 720p video files from different categories on the YouTube website. Using this crawler, we collected more than 22, 000 720p video files from YouTube over 45 days. The average size of the video files is 94 MB, the average duration is 358 seconds and the average bitrate is 2200 Kbps. We randomly chose 500 out of the 22, 000 video files for the video streaming power characterization.

Choice of Smartphones: For power characterization, we chose the Galaxy S, Galaxy S II, and Galaxy Nexus smartphones. Table 1 lists their detailed specifications.

Video Streaming Setup: We stored all the downloaded video files on an internet-accessible HTTP server. We developed a simple Android video player based on the Android Multimedia Framework. This video player reads a list of test videos stored on the HTTP server and automatically plays the video files with a 10-second interval between two consecutive playbacks.

Power Measurement: We measure the power consumed by the smartphones using the Monsoon Power Monitor [3]. The smartphones are powered by the power monitor so they never run out of the battery during the experiments. The power monitor supplies current to the phone and is able to sample the current drawn by the device at a frequency of 5000 Hz. The power consumption data recorded by the power monitor software is time-stamped. The simple android video player also logs when a video playback begins and finishes. With the 10-second interval between playbacks, the offline analysis tool can extract the power consumption data for each video playback from the power trace recorded by the power monitor.

3.1.2 Results

For each smartphone, we measure the average power consumption for playing a video from SD card and from HTTP streaming. Figure 4 shows the average power consumption of the two cases for three smartphones. As shown in the figure, average power consumption by the three smartphones in video streaming is 1.4 W, 1.2 W, and 2.3 W, respectively. Considering the battery capacity of each smartphone, we estimate the battery life of the three smartphones for mobile

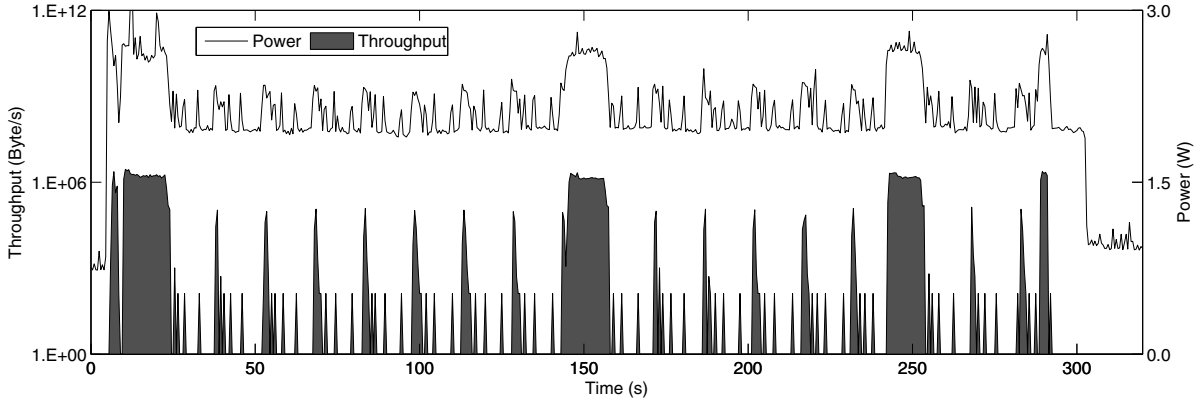

Figure 3: Power trace of 720p HTTP video streaming using Verizon LTE network

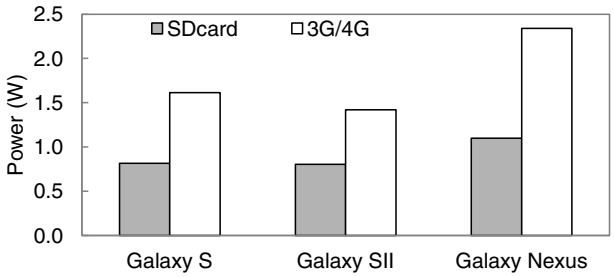

Figure 4: Power consumption characterization

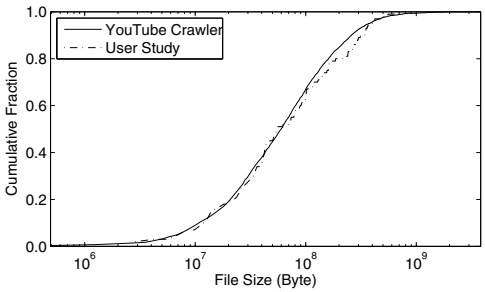

Figure 5: File size distribution of YouTube videos in our experiment

video streaming is 5, 6, and 3 hours, respectively. Moreover, Figure 4 also shows power consumption of video streaming via 3G/4G networks is about 2X of that of playback from SD card. In other words, the 3G/4G radios account for about 50% of the total system power consumption. To summarize, mobile video streaming via 3G/4G network is prohibitively power hungry, and 3G/4G radios are the most significant power consumer in smartphones for video streaming.

A natural question is why 3G/4G radio consumes so much power. We shall now answer this question using the power trace. Figure 3 depicts the transient power consumed by a smartphone (Galaxy Nexus) that is receiving a 2 Mbps video stream from a HTTP server via Verizon LTE network. The figure also shows the corresponding time-stamped data traffic between the HTTP server and the smartphone.

As shown in Figure 3, there are three different data traffic patterns. Each of the four large continuous chunks represents a high-speed downloading session in the *Fetching* state to fill the cache; each of the 15 second interval peaks corresponds to a 64 KB downloading in the Keep-Alive state; and the tiny peaks are caused by TCP zero window probes sent by the HTTP server and the corresponding TCP ACKs as responses from the smartphone. The sequence of data traffic patterns strictly conforms to the state machine described in section 2.3.

Interestingly, the power consumption data shows an almost identical pattern. The highest power consumption happens during the four continuous high-speed downloading, each of the 64 KB downloading slots causes a much lower power consumption peak and the tiny peaks in the data traffic generate several much narrower power consumption peaks. Obviously, there is a 10 second TAIL power state after the last chunk of continuous high-speed downloading which completes the streaming task. After the 10 second TAIL state, the LTE modem enters the IDLE state and the power consumption drops to a dramatically lower level. Given that the interval between 64 KB downloadings is 15 seconds (which is longer than the 10 second TAIL state), it is natural to wonder why the LTE modem never enters the IDLE state before the last continuous downloading. The reason is that the TCP zero window probes and the corresponding TCP ACKs are small packets that effectively bring the LTE modem from the TAIL state back to the ACTIVE state and thus prohibit the LTE modem from staying in the TAIL state long enough.

Thus, excessive power consumption will occur when more than two downloading sessions are needed, i.e., the file size of the video is more than 20 MB. Figure 5 shows the file size distribution of the videos downloaded by our crawler. More than 80% of the 720p videos are larger than 20 MB and thus suffer from excessive power consumption.

3.2 YouTube Usage by Smartphone Users

3.2.1 Methodology

Apparatus: We implemented a logger to record the user inputs in the YouTube app on Android smartphones. Unfortunately, we are unable to modify the YouTube app directly due to its close-source nature. Instead, we implement the logger in the underlying media player of the Android Multimedia Framework. The logger is able to record both video file information and user inputs during a video play-

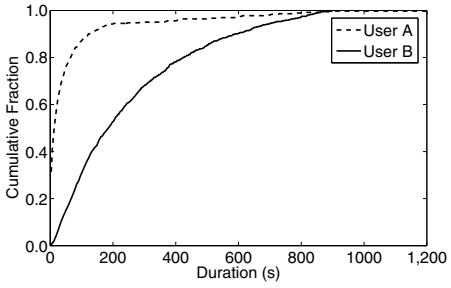

Figure 6: Distribution of user viewing time

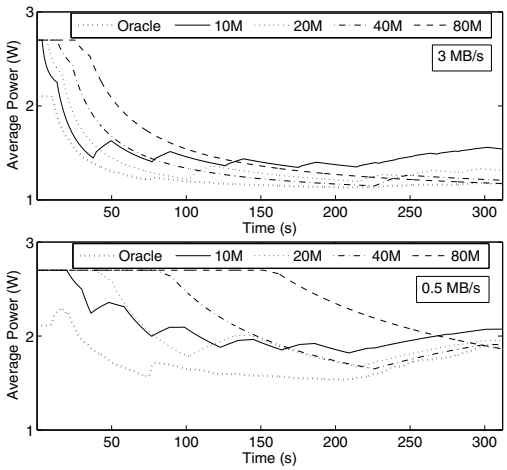

Figure 7: Average power consumption with different cache sizes and different network speeds

back when an app, including YouTube, calls for the service provided the Multimedia Framework. We are particularly interested in the following information: 1) The UID of the app which uses the media player service; 2) The size and the duration of the video file; 3) The time at which the user starts, pauses, resumes, seeks or stops the video playback. Since Android assigns a unique UID for each app, we could filter the log and collect the traces for the YouTube app afterwards.

Subjects: We recruited 10 unpaid participants from Samsung Telecommunications America. Their ages range from 25 to 36. Out of the 10 participants, 4 are female. Every participant is a self-described, heavy, mobile YouTube user who watches at least 10 YouTube videos on his/her smartphone per day, according to our survey.

Procedures: We provided each participant a Galaxy Nexus, with the logger pre-installed, as his/her personal phone for two months. We paid their phone bills for the two months and asked them to use the Verizon LTE network whenever available. Finally, we collected the phones and analyzed the logs after the two month study.

3.2.2 Results

First, most video sessions are short. We define a video session from a start operation to a stop operation. We also treat a forward seek operation as starting a new video session if the seek destination is out of range of the caching content and thus requires a new downloading session.In particular, 80% of YouTube sessions in smartphones are less than half of the corresponding video durations. As a result, downloading the whole video into a large cache and turning off the radio will not solve the problem because of the waste in both data transfer and power. Instead, we should still use multiple download sessions in a video streaming session.

Second, user viewing patterns are different. Figure 6 shows two typical users. For User A, around 80% of the session time is less than 70 seconds. While for User B, over 50% of the session time is larger than 200 seconds. Therefore, we should apply different downloading schedules to different users.

These observations naturally inspired us to explore the power consumption with different cache sizes. We studied this by running simulations with a 75 MB, 312 second long 720p video file. The power consumption is calculated using the Verizon LTE modem power profile measured in section 3.1. We chose four different high threshold values: 10 MB, 20 MB, 40 MB and 80 MB. We also ran an Oracle scheme which knows exactly how much data the user consumes and thus downloads all the data in only one shot.

The top figure of Figure 7 shows the simulation result under 3 MB/s network speed. The bottom one is simulated under 0.5 MB/s network speed and the difference between the two simulations is explained in Section 3.3.2. In this section we focus on the top figure.

The Oracle always has the lowest average power consumption no matter when the user stops the video playback. When the viewing time is short, for example, less than 60 seconds, the average power consumption of a small cache size is better than that of a larger one and the difference is usually non-trivial. With longer viewing time, a choice of large cache size is preferred since it has better average power consumption. Another trend is that as the viewing time grows, the average power consumption difference becomes much smaller, especially for the cache sizes 40 MB and 80 MB.

After observing the different user viewing behaviors, we now propose a better way to manage the cache size in order to reduce power consumption: for users with short viewing durations (User A in Fig. 6), a small cache size is preferred while for users with long viewing durations (User B in Fig. 6), a large cache size is a better choice.

3.3 LTE Network Speed

3.3.1 Methodology

We developed a speed-test app running on the Galaxy Nexus. The app downloads a 100 MB file from a HTTP server every 60 minutes. During the downloading, the app measures the average speed of each 5-second period and logs the speed to a disk file.

3.3.2 Results

We ran the speed-test app for a week and analyzed its log file. Figure 8 shows the result for a typical day. As shown in the figure, the average network speed measured at each hour varies dramatically throughout the day. We observe higher speed during night when the number of active users are much less. Most importantly, the network speed varies sharply even within a short period. For example, at 8PM, the highest speed and lowest speed is 4 MB/s and 1 MB/s, or 4X difference.

To study the impact of network speed variation on power

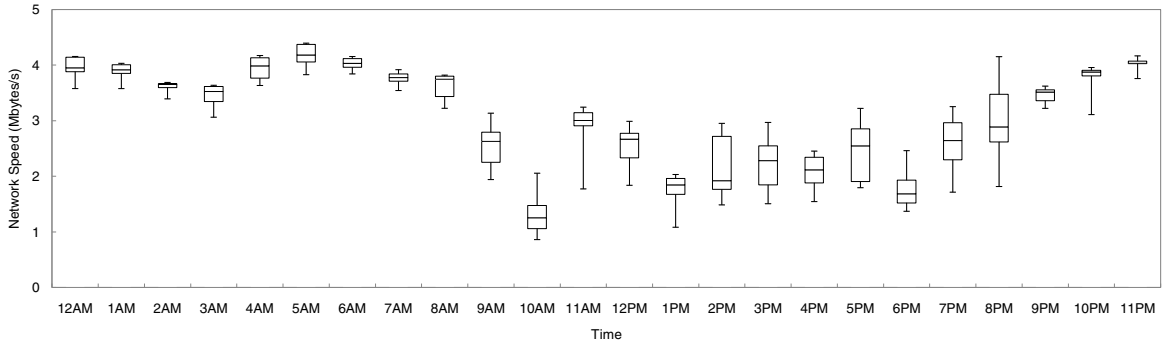

Figure 8: Verizon LTE network speed

consumption, we performed a simulation similar to that described in section 3.2, but with different network speed settings. As shown in Figure 7, the network speeds for the top figure and the bottom figure are 3 MB/s and 0.5 MB/s respectively. The average power consumption with 0.5 MB/s is higher for all cache sizes. The power consumption difference between different cache sizes are much larger and the curves converge much slower the curves with the higher network speed. In observance of such obvious impact of network speed on power consumption, we must adapt the cache size to real time network speed.

4. GREENTUBE DESIGN

We now describe the design of GreenTube as motivated by the results from the preceding studies.

4.1 Key Design Decisions

The results from the preceding motivational studies lead us to make the following design decisions for GreenTube.

Close the TCP connection when the cache is full. To reduce excessive power consumption as illustrated in Section 2.1, GreenTube disconnects from the HTTP server after each downloading session when the cache is full. Note that there are many other options to disconnect from the HTTP server. For example, we could have turned off the 3G/4G radio and released the IP address. However, we chose to close the TCP connection for two reasons. First, both alternatives need to wait for the base station to re-allocate an IP address when a new downloading session needs to start. This re-allocation process can take as long as 5 seconds in Verizon LTE network according to our measurements. In contrast, to re-establish a new TCP connection with the HTTP server requires less than 0.2 seconds. This 25x delay reduction significantly decreases the risk of streaming interruption and excessive power wastage. Second, unlike turning off the radio or releasing the IP address, closing the TCP connection will not affect background services that may require network connection, such as email checking and data synchronization. Actually, such TCP disconnection feature has also been adopted by the YouTube app that is shipped with the Android system, as described in Section 2.3. GreenTube improves over the YouTube app in the following aspects.

Record user-specific viewing history. GreenTube keeps a record of the user viewing time for each streaming session and utilizes this historical record to generate a probability distribution for the user viewing time. Such a probability distribution, as demonstrated in Section 2.2, has a significant impact on average power consumption of the streaming session. At the beginning of each downloading session, GreenTube uses the probability distribution to estimate the expected viewing time for the user and chooses the optimal cache size that leads to minimal expected power consumption. Note that GreenTube treats forward seek operation as starting a new video session if the seek destination is out of range of the caching content and thus requires a new downloading session.

Adapt cache size to network speed. GreenTube samples network speed in each downloading session. As discussed in Section 2.3, network speed has a huge impact on power consumption and varies dramatically over a short period. As a result, the optimal cache size should adapt to the network in real time. In each downloading session, GreenTube estimates the network speed for every second using the downloaded data size from the previous second. Based on this estimation, GreenTube adjusts the cache size accordingly.

Set maximal cache size based on user's choice. GreenTube chooses cache size from a fixed set. The maximal cache size is equal to the worst-case amount of excessive downloaded data in a streaming session and thus should be determined by the user based on his/her tolerance for data wastage. Besides the user's choice, GreenTube also sets a cache-size upper limit because the marginal power saving will diminish as cache-size increases. To quantitatively decide such a limit, we measured the average power consumption by Galaxy Nexus for video streaming via Verizon LTE network using different cache sizes. Given a cache size, we used it for all the test videos and then compiled Figure 9 to show the average power consumption corresponding to each cache size. As illustrated in the figure, the marginal power saving is negligible when cache size is larger than 80 MB. Therefore, GreenTube sets the maximum cache size to be the lower of 80 MB and the user's choice.

To summarize, GreenTube fetches a video file from the HTTP server in multiple downloading sessions and disconnects from the server after each downloading session ends. The starting and ending time of each downloading session (except for the starting time of the first session and ending time of the last session) is determined by cache size. GreenTube judiciously schedules downloading sessions by adaptively adjusting the cache size according to user viewing history and real-time network speed. We call this adaptive adjustment process *Dynamic Cache Management*.

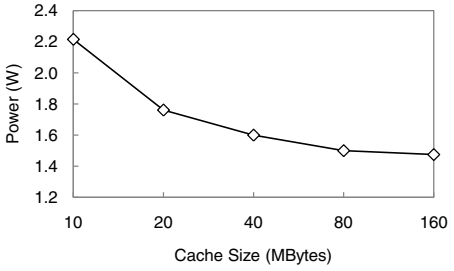

Figure 9: Power consumption by Galaxy Nexus for video streaming via Verizon LTE network using different cache sizes

4.2 Dynamic Cache Management

We next present the *Dynamic Cache Management* (DCM) algorithm. The basic idea is to adaptively adjust the high threshold value of the cache based on the sampled network speed and the expected viewing time calculated from the probability distribution for the user's viewing time. After each second has elapsed, the algorithm computes the high threshold value that correponds to minimal energy consumption assuming that the user will stop viewing at the expected viewing time.

Algorithm 1 shows pseudocode for the DCM algorithm. th_h and th_l are the high threshold value and low threshold value respectively. Initially the high threshold value is set to 10 MB while the low threshold value always remains fixed at 4 MB. D is the video duration while A is the size. This information is extracted from the MP4 header after the header has downloaded completely. $S_{history}$ is the discrete probability distribution calculated from the user viewing history and it is normalized. Every time the user watches a new video, the data is added to $S_{history}$. B is the set of candidate high threshold values. P_{active}, P_{tail}, P_{idle} and T_{tail} are power profile related power consumption parameters derived from our measurement. Specifically for the Verizon Galaxy Nexus, P_{active} is the average power consumption when the LTE modem is in the ACTIVE state, P_{tail} is the average power consumption in the TAIL state, P_{idle} is the average power consumption in the IDLE state and T_{tail} is the maximum time the LTE modem stays in the TAIL state before it transits into the IDLE state.

The DCM algorithm is event-driven. It adjusts th_h every second during a continuous, active downloading. There is no need to adjust th_h in the *Idle* state because it will have absolutely no impact on the downloading behaviour. The adjustment is done in two steps: 1) Calculate the expected viewing time using function EXPECTED-VIEWING-TIME; 2) Using the expected viewing time, estimate the energy consumption for each high threshold candidate and thus determine the optimal th_h.

We obtain the energy-consumption estimate by simulating the periodic behaviour observed in our video playback power measurement. Given the current time t_{cur}, the expected viewing time t_{exp}, the current network speed s, the size of the video A, the size of the consumed video content a_{cur}, the duration of the video D and the various power-profile parameters (P_{active}, P_{tail}, P_{idle}, and T_{tail}), the function OPTIMAL-CACHE-SIZE first estimates the time spent in the ACTIVE, TAIL and IDLE states respectively in the time period from t_{cur} to t_{exp}. It then uses this time estimate to derive the energy consumption.

Algorithm 1 The DCM Algorithm

```
Wait for event
if event is downloading started then
    Set timer to expire in 1 second
else if event is timer expired then
    t_exp = EXPECTED-VIEWING-TIME(t_cur)
    x_opt = OPTIMAL-CACHE-SIZE(t_exp, t_cur)
    th_h = x_opt
    Set timer to expire in 1 second
else if event is downloading stopped then
    Cancel the timer
end if

function EXPECTED-VIEWING-TIME(t_cur)
    Ŝ = {p_i | p_i ∈ S_history, i · D/|S_history| ≥ t_cur}
    S'_history = normalize(Ŝ)
    n = |S'_history|
    t_exp = Σ_{i=1}^{n} p_i · (t_cur + (i − 1) · D/n), p_i ∈ S'_history
    return t_exp
end function

function OPTIMAL-CACHE-SIZE(t_exp, t_cur)
    s = current downloading speed
    Set bitrate r = (A − a_cur)/(D − t_cur)
    e_opt = +∞
    x_opt = 0
    for each b in B do
        t = t_cur
        t_active = 0
        t_idle = 0
        while t < t_exp do
            t_d = min(t_exp − t, (b − th_l)/s)
            t_d = min((A − a_cur − th_l)/s, t_d)
            a_cur = a_cur + s · t_d
            t_active = t_active + t_d
            if a_cur + th_l ≥ A then
                t_v = (s·t_d + th_l)/r
            else
                t_v = s·t_d/r
            end if
            t_v = min(t_v, t_exp − t)
            t_idle = t_idle + min(t_v − t_d, T_tail)
            t = t + t_v
        end while
        t_idle = (t_exp − t_cur) − t_active − t_tail
        e = P_active × t_active + P_tail × t_tail + P_idle × t_idle
        if e < e_opt then
            e_opt = e
            x_opt = b
        end if
    end for
    return x_opt
end function
```

5. EVALUATION

In this section, we present the implementation of GreenTube and evaluate its performance.

5.1 Implementation

To evaluate the performance of GreenTube, we implemented a C++ prototype based on the Android Multimedia Framework. We created a new data-source provider that wraps the default HTTP streaming data-source provider and incorporates our dynamic cache management algorithm.

Configuration settings allow Android apps to decide at runtime whether to use our dynamic cache management. By doing so, the impact to the existing code is minimum and the modification is transparent to the Android apps. The implementation changes 6 source files around 700 lines of code. According to our measurements, the computational cost of our implementation is negligible. On average, our DCM algorithm consumes less than 100 microseconds on the Galaxy Nexus.

5.2 Evaluation Methodology

As mentioned in section 3.2, we conducted a user study with the popular YouTube app and obtained 4000 video playback traces over a period of 2 months with 10 users. By analyzing the traces, we obtained the URLs of all the videos and downloaded them from YouTube. We stored these videos on an Internet-accessible HTTP server. The videos and the corresponding traces were used to evaluate GreenTube.

We implemented a simple media player which 1) uses the dynamic cache management enabled data source provider, and 2) is able to read the user study trace, set the video source to the user watched video stored on our HTTP server and mimic the recorded user viewing behavior. To run the experiment automatically without user intervention, the media player also reads a play list containing ordered sequence of video files for testing. To automatically synchronize the video playback and the power trace, we again leverage the logger facility in the Android Multimedia Framework to log the time when each video playback starts and stops. Also, there is a 10-second interval between two consecutive video playbacks, to enable our offline analysis tool to easily extract the power traces for individual video playbacks.

The videos watched by each user were randomly divided into 8 groups. In round-robin fashion, each one of the 8 groups of videos is chosen as the test set and the traces of the remaining 7 groups were used only for prediction. By iteratively choosing each of the 8 groups as the testing group, all videos watched by the users were covered.

We compare the following four schemes:

- **Android:** The default Android scheme which is discussed in section 2.3. It usually produced the highest power consumption.
- **YouTube:** The scheme used by the YouTube app, which closes the TCP connection every time the cache is full. This greatly increases the time the LTE modem stays in the IDLE state so the power consumption can be substantially improved.
- **DCM:** Our DCM scheme, which is based on the YouTube scheme but changes the cache size dynamically according to both user behavior and network speed.
- **Oracle:** The Oracle scheme, which always knows when the user stops the video playback and downloads the exact amount of data. Such scheme, although impossible to implement practically, is included as the optimal bound, i.e., the lowest power consumption that can be achieved.

5.3 Performance Results

We now present evaluation results from measurements and simulations.

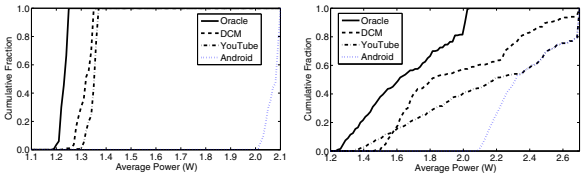

Figure 10: Power consumption distribution with Verizon LTE network for different users

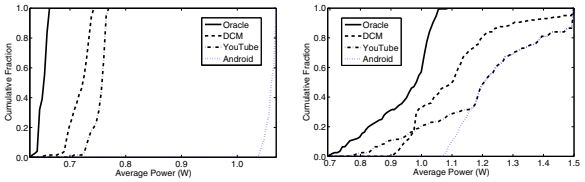

Figure 11: Power consumption distribution with ATT HSPA network for different users

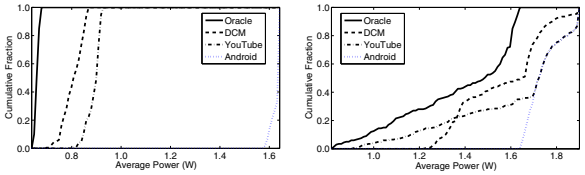

Figure 12: Power consumption distribution with T-Mobile HSPA+ network for different users

5.3.1 Overall Power Reduction

Figure 10, 11 and 12 show the power consumption distribution of the four schemes on Verizon LTE network, ATT HSPA network and T-Mobile HSPA+ network respectively.

Figures on the left in each of the three pairs of figures depict users (for example, User B) who tend to finish viewing the videos. Figures on the right depict users (for example, User A) who usually watch only the very beginning portion of the video.

For all networks, when the user tends to finish the video, our DCM scheme clearly consumes much less power than the YouTube and the Android schemes. The Android scheme has the worst power consumption and the the performance gap is huge. Although both the DCM and the YouTube schemes use the "disconnect when cache is full" feature, our DCM scheme clearly outperforms the YouTube scheme because of the accurate viewing-time prediction and the dynamic selection of a larger cache.

For users who stop viewing a video very early, our DCM scheme is much better than the Android scheme. Comparing against the YouTube scheme, the DCM scheme performs slightly worse in a few cases while in other cases, the advantage is much more obvious than for the other type of users. Our analysis of the traces shows that when the predicted viewing time is larger than the actual viewing time, our DCM scheme tends to choose a cache size larger than 20 MB which is used in the YouTube scheme. Thus our DCM scheme downloads more data and consumes more energy than the YouTube scheme. For other cases, our DCM scheme tends to choose a smaller cache size and the power saving is substantial.

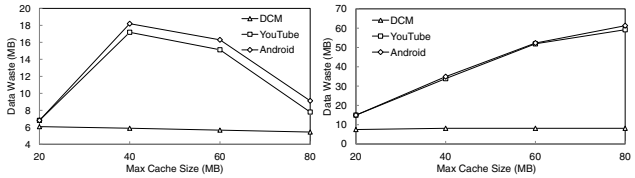

Figure 13: Data wastage with different maximum cache sizes for different users

5.3.2 Impact on Data Wastage

To evaluate each scheme's impact on data wastage, we collected the size of wasted data, i.e., the video data which is downloaded but not viewed by the user. We excluded the Oracle from this analysis because it never wastes data. We ran a series of simulations for the other three schemes with the following cache sizes: 20 MB, 40 MB, 60 MB and 80 MB. For the DCM scheme, these values are the maximum cache sizes.

The simulation results in Figure 13 show the data wastage for users who tend to finish viewing videos. For the YouTube and the Android schemes, the trend is almost the same. When the cache size increases, the data wastage for the two schemes first increases and then decreases. This trend is caused by the mismatch of the data downloaded in the periodic fashion and the data actually consumed by the user. The YouTube and the Android schemes waste the same amount of data at the cache size 20 MB and the Android scheme wastes slightly more data with larger cache sizes. This small difference is caused by the 64-KB data chunk downloading between two consecutive *Fetching* state instances. This also applies to Figure 13(b). Our DCM scheme performs very well regardless of the maximum cache size and the data wastage is almost constant. The reduction in data wastage can be as high as 60%.

Figure 13(b) shows the data wastage for users who stop viewing a video very early. Again, our DCM scheme performs very well and the data wastage is almost constant. Since the user only watches the video for several tens of seconds and our DCM scheme can capture and predict this behaviour, cache sizes smaller than 20 MB are preferred. As a result, the maximum cache size larger than 20 MB has almost no impact on the data wastage. For the YouTube and the Android schemes, the data wastage increases drastically when the cache size increases. Since the user only watches the video for a short duration and the network speed is much higher than the video bitrate, even a cache size of 20 MB is larger than the average size of the viewed video content, the data wastage under these two schemes increases monotonically.

6. RELATED WORK

Wireless interface power optimization: Many pioneering works have been performed to study the air-interface power consumption and optimization. For instance, uPM [11] allows WiFi state adjustment at micro seconds level, Catnap [4] schedules the TCP transfers based on the granularity of the application data units to put WiFi interface into longer sleep. Mohapatra *et al.* [15] proposed an integrated power management approach. Among them, network traffic regulation is closely related to our work. It proposes to use additional buffer at Access Point (AP), then stream data can be transmitted to the client using burst rate. After the burst, the WiFi interface can be switched into a sleep state. In our work, we studied the energy consumption for WiFi, 3G and 4G LTE in video streaming scenario. Our proposal can be combined with these proposed optimization scheme to further reduce the interface energy.

Video streaming: Efficient video streaming over a network has been studied extensively for many years. For example, peer-to-peer (P2P) streaming has been proposed and deployed successfully [22, 10]. More studies have been carried out to improve the existing P2P system, such as using machine learning [16], using hybrid P2P structure [21], and using layered video [14]. Note that almost all these works try to improve the network utility or throughput for video streaming. However, none of them considers the energy-constrained mobile streaming scenario, which is the other important topic addressed extensively in the literature. For instance, Xiao *et al* [20] performed power-consumption measurement for YouTube video streaming on Nokia S60 platform over WCDMA and WLAN networks. However, their power-consumption measurements will be quite different from those in our work because of significant recent SmartPhone hardware advances. Later Mohamed *et al* [8] proposed to transmit the video using bursts on DVB-H system such that mobile devices can receive a burst of traffic and then turn off their radio frequency circuits till the next burst in order to save energy. However, it is inefficient to turn off the radio circuits for 3G and 4G/LTE since it requires much more power for re-connection. He *et al.* [7] proposed a parametric power-rate-distortion model for H.263 based video encoding, and wireless streaming. This work can be combined with our solution. Moreover, both CoolSpots [17] and BlueStreaming [13] use BlueTooth to offload delay sensitive control traffic (which is about twice as much as streaming data traffic based on Internet measurement [12]) while keeping WiFi for data traffic streaming in P2P video streaming scenario. BlueTooth interface is always active, however the power consumption overhead can be over-compensated by putting WiFi interface into power saving mode (since WiFi does not need to transmit control traffic actively). In general, our work is complementary to CoolSpots and BlueStreaming.

7. DISCUSSION

We next discuss the limitation of our work and future direction.

7.1 Display Power in Video Streaming

The focus of this paper is to optimize the energy efficiency of 3G/4G radios that contribute to over 50% of system power consumption of a smartphone. However, other hardware components are also worth investigating. Many researchers have identified the display, especially OLED display, as a major power consumer [6] [5]. Fortunately, we found that the OLED display power consumption in video streaming is not as significant as in other applications, such as web browsing [5]. We arrived this conclusion after performing the following experiments. First, we analyzed the RGB value distribution from the video files downloaded from YouTube. Then, we estimated the average power consumption by OLED display to play these videos using the OLED

power model described in [5]. Our result shows that the OLED display power consumption on Galaxy S, Galaxy S II, and Galaxy Nexus is 190 mW, 200 mW, and 150 mW, respectively. Such low power consumption is due to the relatively low RGB values in real-world videos.

7.2 Network Support

As discussed in this paper, the fundamental reason why 3G/4G radio consumes so much power is the presence of relatively long TAIL state. Therefore, an alternative and effective solution to reduce 3G/4G radio power is to make the smartphone notify the base station and initialize the power state demotion after completion of each downloading session. By doing this, the smartphone can immediately set the interface into IDLE state instead of waiting for the expiration of the tail timer. As aforementioned, such a TAIL state consumes a large portion of the power as well. By eliminating the tailstate power, the 3G/4G radio can be very power efficient. This type of feedback channel enabling between mobile client and base station is worthy of further study and standardization along with the process of 3GPP LTE and LTE-Advanced protocols.

8. CONCLUSION

The power consumption of a 3G/4G radio is a significant drain on the battery of a smartphone and thus severely hurts the user experience of mobile video streaming. We show that such high power consumption is due to the conflict between HTTP streaming support in state-of-the-art smartphone systems and power state transition defined in 3G/4G networks. To resolve this conflict, we present the design and realization of GreenTube, a light-weight software solution that is readily deployable on off-the-shelf Android-based smartphones. GreenTube employs a novel, dynamic cache management algorithm that judiciously schedules downloading sessions to reduce average power consumption for mobile video streaming. Our results show that GreenTube achieves up to 70% and 40% power reduction for 3G/4G radio and the whole smartphone system, respectively. Besides power reduction, GreenTube also reduces data wastage by 60%. Because of its superior performance, we believe GreenTube is worth considering as a revision to the current Android system.

9. REFERENCES

[1] `http://www.techweekeurope.co.uk/news/lte-power-consumption-triggers-battery-worry-61769/`.

[2] `http://www.allot.com/MobileTrends_Report_H2_2011.html?campid=701D000000012aR/`.

[3] `http://www.msoon.com/LabEquipment/PowerMonitor/`.

[4] Dogar, F. R., Steenkiste, P., and Papagiannaki, K. Catnap: exploiting high bandwidth wireless interfaces to save energy for mobile devices. In *Proc. of the 8th international conference on Mobile systems, applications, and services* (2010), pp. 107–122.

[5] Dong, M., and Zhong, L. Chameleon: a color-adaptive web browser for mobile oled displays. In *Proc. of the 9th international conference on Mobile systems, applications, and services* (2011), pp. 85–98.

[6] Falaki, H., Mahajan, R., Kandula, S., Lymberopoulos, D., Govindan, R., and Estrin, D. Diversity in smartphone usage. In *Proc. of the 8th international conference on Mobile systems, applications, and services* (2010), pp. 179–194.

[7] He, Z., Liang, Y., Chen, L., Ahmad, I., and Wu, D. Power-rate-distortion analysis for wireless video communication under energy constraints. *IEEE Trans. on Circ. and Sys. for Video Tech. 15*, 5 (May 2005), 645 – 658.

[8] Hefeeda, M., and Hsu, C.-H. On burst transmission scheduling in mobile tv broadcast networks. *IEEE/ACM Trans. Netw. 18*, 2 (Apr. 2010), 610–623.

[9] Huang, J., Qian, F., Gerber, A., Mao, Z. M., Sen, S., and Spatscheck, O. A close examination of performance and power characteristics of 4G LTE networks. to appear ACM MobiSys '12.

[10] Huang, Y., Fu, T. Z., Chiu, D.-M., Lui, J. C., and Huang, C. Challenges, design and analysis of a large-scale p2p-vod system. In *Proc. of the ACM SIGCOMM 2008 conference on Data communication* (2008), pp. 375–388.

[11] Liu, J., and Zhong, L. Micro power management of active 802.11 interfaces. In *MobiSys* (2008), pp. 146–159.

[12] Liu, Y., Guo, L., Li, F., and Chen, S. An empirical evaluation of battery power consumption for streaming data transmission to mobile devices. In *Proc. of the 19th ACM international conference on Multimedia* (2011), pp. 473–482.

[13] Liu, Y., Li, F., Guo, L., Guo, Y., and Chen, S. Bluestreaming: towards power-efficient internet p2p streaming to mobile devices. In *Proc. of the 19th ACM international conference on Multimedia* (2011), pp. 193–202.

[14] Liu, Z., Shen, Y., Ross, K. W., Panwar, S. S., and Wang, Y. Layerp2p: using layered video chunks in p2p live streaming. *IEEE Trans. Multi. 11*, 7 (Nov. 2009), 1340–1352.

[15] Mohapatra, S., Cornea, R., Dutt, N., Nicolau, A., and Venkatasubramanian, N. Integrated power management for video streaming to mobile handheld devices. In *Proc. of the 11th ACM international conference on Multimedia* (2003), pp. 582–591.

[16] Niu, D., Li, B., and Zhao, S. Self-diagnostic peer-assisted video streaming through a learning framework. In *Proc. of the ACM international conference on Multimedia* (2010), pp. 73–82.

[17] Pering, T., Agarwal, Y., Gupta, R., and Want, R. Coolspots: reducing the power consumption of wireless mobile devices with multiple radio interfaces. In *Proceedings of the 4th international conference on Mobile systems, applications and services* (2006), pp. 220–232.

[18] Qian, F., Wang, Z., Gerber, A., Mao, Z., Sen, S., and Spatscheck, O. Profiling resource usage for mobile applications: a cross-layer approach. In *Proc. of the 9th international conference on Mobile systems, applications, and services* (2011), pp. 321–334.

[19] Shepard, C., Rahmati, A., Tossell, C., Zhong, L., and Kortum, P. Livelab: measuring wireless networks and smartphone users in the field. *SIGMETRICS Perform. Eval. Rev. 38*, 3 (Jan 2011), 15–20.

[20] Xiao, Y., Kalyanaraman, R. S., and Yla-Jaaski, A. Energy consumption of mobile youtube: Quantitative measurement and analysis. In *Proc. of the 2nd International Conference on Next Generation Mobile Applications, Services, and Technologies* (2008), pp. 61–69.

[21] Yin, H., Liu, X., Zhan, T., Sekar, V., Qiu, F., Lin, C., Zhang, H., and Li, B. Design and deployment of a hybrid CDN-P2P system for live video streaming: experiences with livesky. In *Proc. of the 17th ACM international conference on Multimedia* (2009), pp. 25–34.

[22] Zhang, X., Liu, J., Li, B., and Yum, T.-S. P. Coolstreaming/donet: a data-driven overlay network for peer-to-peer live media streaming. In *Proc. of IEEE INFOCOM* (2005), pp. 2102–2111.