Using Crowd-Sourced Viewing Statistics to Save Energy in Wireless Video Streaming

Mohammad Ashraful Hoque, Matti Siekkinen, Jukka K. Nurminen
Aalto University School of Science, Finland
{mohammad.hoque, matti.siekkinen, jukka.k.nurminen}@aalto.fi

ABSTRACT

Video streaming on smartphones is one of the most popular but also most energy hungry services today. Using mobile video services results in two contradictory sources of energy waste for smartphones: i) energy waste because of excessively aggressive prefetching of content that the user will not watch because of abandoning the session, and ii) excessive amount of tail energy, which is energy wasted by keeping the wireless interface powered on after receiving a chunk of content; this is caused by prefetching chunks that are too small. To remedy this, we propose a novel download scheduling algorithm based on crowd-sourced video viewing statistics. Our algorithm judiciously evaluates the probability of a user interrupting a video viewing in order to perform the right amount of prefetching. In this way, the algorithm balances the amount of the two above-mentioned kinds of energy waste. By simulations, we show that our scheduler cuts the energy waste to half compared to existing download strategies. We have also developed an Android prototype that implements the download scheduler together with a novel downloader that speeds up the download by exploiting the Fast Start technique. The prototype exhibits the desired properties of the scheduler, and its faster downloading mechanism yields further energy savings of up to 80% compared to the default Android YouTube app.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems[Distributed Application, Client/Server]

Keywords

Energy Saving; Mobile Multimedia; Tail Energy

1. INTRODUCTION

Energy consumption in modern smartphones has a significant impact on user satisfaction. Radio communication using the device's wireless network interfaces, such as Wi-Fi, 3G, and LTE, consumes a significantly large fraction of the total energy. This energy consumption is specifically of concern for services that are frequently used and that generate a lot of traffic. Mobile video streaming is a prime example of such services.

Today, mobile video services employ a range of techniques to deliver content to smartphones. The delivery mechanisms differ in the rate that the content is transmitted and in the amount of content prefetched and buffered by the player. The chosen technique depends on the behaviour of both the server and the client application [21, 16]. An aggressive prefetching technique improves user experience of watching a video but may lead to significant amount of unnecessarily downloaded content if the user abandons watching the video [11]. As a result, the associated downloading energy is also wasted. On the other hand, downloading the video content in small chunks reduces the probability of downloading unnecessary content but it is also energy inefficient because of the so-called *tail energy* present in mobile wireless communication. This energy is consumed as a result of keeping the radio powered on for a while after receiving or transmitting the last bit of content.

This paper presents a novel download scheduling algorithm which optimizes the energy consumption while streaming video to a mobile device via a wireless network interface (WNI). The novelty lies in the algorithm leveraging crowd-sourced video viewing statistics in order to quantify the probability of a user abandoning the watching of a video. The algorithm then calculates an energy optimal download schedule based on these probabilities. Extensive evaluations of our algorithm show that such statistics help to cut the energy waste to half compared to the best known alternative download strategies.

In summary, this paper makes the following contributions.

- We identify the sources of energy inefficiency in mobile video streaming and demonstrate that current download strategies suffer from these problems. Specifically, we formulate the problem of energy optimization of download scheduling in video streaming when users have a non-zero probability of abandoning the session.

- We design eSchedule, which is a download scheduling algorithm based on crowd-sourced video viewing statistics and power modeling. We show through simulations that this algorithm is able to cut the energy waste to half compared to currently existing other approaches.

MobiCom'13, September 30 - October 04 2013, Miami, FL, USA

http://dx.doi.org/10.1145/2500423.2500427.

- We present a YouTube-compliant Android prototype called StreamThrottler, which comprises two components: implementation of eSchedule and a novel downloader that downloads an individual chunk of content at the maximum bulk transfer capacity. We evaluate it on Samsung Galaxy SIII 4G and compare the performance of StreamThrottler to the default Android YouTube player.

The rest of the paper is structured as follows. The next section outlines the background to mobile video streaming, energy consumption and lists the factors that result in energy being wasted. Section 3 presents our novel download scheduler, called eSchedule, which we evaluate by applying simulations, in Section 4. Section 5 describes and evaluates the Android prototype called StreamThrottler. In Section 6, we discuss the practical issues associated with the player buffer, bandwidth fluctuation and audience retention information. Finally, we contrast our work with earlier research in section 7 before concluding the paper.

2. WIRELESS VIDEO STREAMING AND ENERGY CONSUMPTION

In this section, we discuss the different forms of energy waste that emerge during streaming video over wireless technologies. The focus is HTTP-based streaming over TCP because of its dominant status. We first briefly consider the different ways that streaming traffic is delivered from the server to the client and show the resulting energy consumption in each case. Then, we explicitly highlight the sources of energy waste that motivate us to propose the novel download scheduling scheme described in Section 3.

2.1 Video Streaming using HTTP over TCP

Today HTTP over TCP is by far the most commonly used protocol suite exploited by streaming services to deliver mobile videos [13]. The streaming services have to accommodate bandwidth fluctuation and jitter, which commonly occur due to the best-effort service of the Internet. To remedy these, content is prefetched from the streaming server and stored at the client player's playback buffer. This buffering is performed right at the beginning of the streaming session and, for this reason, it is commonly referred to as Fast Start.

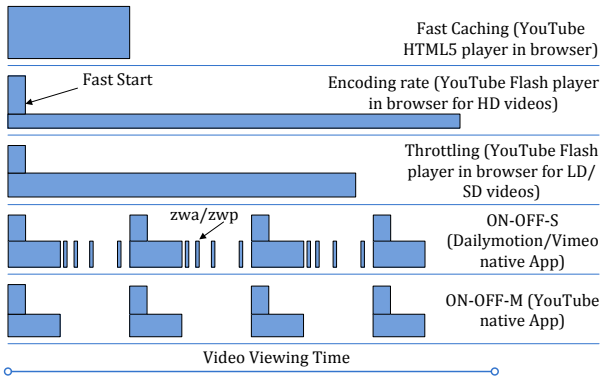


Figure 1: Typical mobile video streaming strategies with different mobile video services. HD refers to high definition videos (e.g., 720-1080p). LD and SD refer to low and standard definition videos respectively (e.g., 240-480p). The height of a rectangle represents the download rate and the width the download time.

Following the Fast Start buffering, the technique for delivering the remaining content varies and depends on the combination of the service, player implementation, and device characteristics. From the traffic pattern of YouTube, Dailymotion and Vimeo players in Samsung Galaxy SIII 4G (Android Jelly Bean 4.1.2) we identified a number of strategies employed by these video services for constant bit rate streaming (see Figure 1). YouTube exhibits all the streaming techniques in the Android platform with its combination of browser and the native players, except ON-OFF-S. The Dailymotion, and Vimeo applications apply ON-OFF-S.

Figure 1 shows large differences occur in the way the wireless interface is utilized, depending on the type of strategy. Fast Caching simply downloads the whole video in one go and the wireless interface is efficiently utilized. However, because users often do not watch the whole video, using this strategy may lead to a significant amount of content being unnecessarily downloaded [11].

Encoding rate streaming results from a situation where the server intends to send initially more data than the client player is able or willing to buffer. As a consequence, TCP flow control activates and allows the server to send the rest of the content at the rate at which the client player consumes it, i.e. encoding rate. This results in a light but continuous utilization of the wireless interface.

By throttling the sending rate, the server sends content to the client at a rate which is lower than the bulk transfer capacity but higher than the stream encoding rate. The YouTube Flash player in browser always specifies a throttling rate of 1.25 in the HTTP request and the server sends content at 1.25 times the encoding rate to the client.

The ON-OFF strategies are governed exclusively by the client players. There are two types: In the first, ON-OFF-M, the client player establishes a new TCP connection and makes a new HTTP request for each ON period and closes the connection after having received a chunk of content. In the second, ON-OFF-S, the client player uses a persistent TCP connection and simply stops reading from the TCP socket during an OFF period. Because the server has more data to send, TCP flow control messages, i.e. zero window probes (zwp) from the server and advertisements (zwa) from client, are exchanged during the OFF period. The YouTube player in Galaxy SIII buffers 100 s worth of content. When the buffer drains to 40 s, the player resumes downloading. Therefore, the duration of an OFF period is 60 s. The Dailymotion and Vimeo players buffer 20 MB of content. Figure 1 also shows that the players receive content at some throttled rate after the Fast Start phase using both ON-OFF mechanisms. In a manner similar to ON-OFF-M, GreenTube [16] also uses multiple TCP connections to download a video. It uses local viewing history of a user to determine the playback buffer size which in turn limits the amount of data to be downloaded during each connection.

The selection of the described streaming techniques does not depend on the wireless interface being used for streaming. The selection does not depend on the available bandwidth either. The ON-OFF mechanisms and throttling persist as long as the bandwidth is equivalent to the throttling rate. If the bandwidth is reduced further, the duration of an ON period increases using either of the ON-OFF mechanisms. However, Figure 1 illustrates that the selection depends on the service, player type, and the quality. A detailed study on how the video streaming services exhibit different streaming techniques can be found in [14].

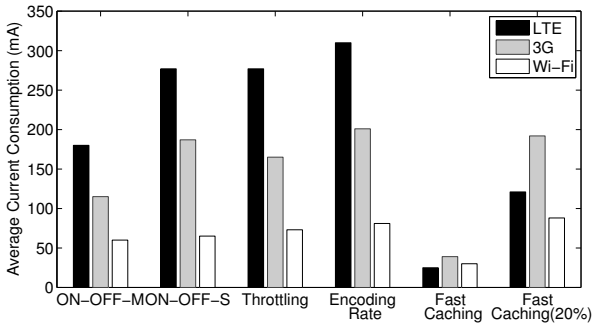


Figure 2: Avg. current consumed by Wi-Fi, 3G and LTE interfaces during streaming sessions of a 597 s video to Galaxy SIII 4G using different streaming techniques.

2.2 Energy Consumption

With regard to energy consumption, all the wireless network interfaces have a few characteristics in common. All of them have mechanisms to ensure that the radio is not kept fully powered on all the time. Consequently, they operate in different modes, which also map to different power states. For example, the states in Wi-Fi that employ a power saving mode (PSM) [3] are receive, transmit, idle, and sleep, of which the first three modes draw, by an order of magnitude of one, more power than the fourth mode. In 3G, the modes correspond to the different channel allocations (CELL_DCH, CELL_FACH, CELL_PCH and IDLE) [2]. An LTE device can be either connected or in idle mode and, if DRX is activated, the radio is switched on only for a short period, otherwise it remains in sleep mode during the connected mode [4].

Another common feature is that transitions from active to more passive states, i.e. decisions about switching the radio to sleep mode, are executed based on inactivity timers. The timer values of 3G and LTE may extend to ten seconds, whereas Wi-Fi switches the radio into sleep mode after only two hundred milliseconds. The timers in cellular networks are not specified in the standards but are set and controlled by the network operators. The energy consumed by keeping the radio on for the period specified by the inactivity timer is often called *tail energy* [10]. This tail energy is the price to pay for more efficient radio access network resource consumption and shorter application-level latency especially when using sporadically communicating applications. Modern smartphones try to avoid long tail energy by employing a different standard, namely Fast Dormancy (FD) [5]. The mobile devices use a shorter inactivity timer of 3-5 s [8]. FD enables a mobile device to switch directly from CELL_DCH to CELL_PCH or to IDLE state.

Using three video streaming services, YouTube, Dailymotion, and Vimeo, we executed several sessions on an Android phone, the Samsung Galaxy SIII 4G. For each streaming technique we conducted three sessions, each for a different WNI (Wi-Fi/3G/LTE). To understand what the different streaming strategies mean from an energy consumption point of view, we performed power measurements using the Monsoon Power Monitor [1]. We discovered that the smartphone used FD in the 3G network and the value of the inactivity timer was 5 seconds. However, the LTE network did not support DRX and the inactivity timer was of 10 seconds.

The total current measured during streaming comprises the idle current, the current draw required by playback and the current draw generated by the downloading, i.e. wireless interface usage. We calculated the playback plus idle current from the power traces by manually isolating a period following completion of the stream download and when only playback remains. We then computed the downloading current by subtracting the playback current from the total current. Figure 2 shows the computed downloading current. We observed that the playback plus idle current (not shown in the figure) varied 200-230 mA, which highlights the fact that the energy spent for downloading only is in most cases a very significant part of the total energy.

Among the different strategies we examined, it is clear that encoding rate streaming consumes the most energy because it keeps the radio active almost all the time. This is because the inactivity timers do not expire in between packet receptions. The phone consumes less current than the encoding rate when the server throttles the bit rate, because the device finishes downloading a little bit earlier. ON-OFF-S also leads to larger energy consumption when used over LTE and 3G because their long inactivity timer values combined with the TCP flow control traffic keeps the radio on most of the time. By contrast, ON-OFF-M is much more energy efficient. Compared to the discussed methods, Fast Caching draws very little current on average while the whole video is being watched. In Figure 2, we also present a case where the user abandons the session after watching 20% of the video, in which case the average current is clearly higher. The reason being that in the beginning the device does unnecessary work by downloading content that is never watched.

2.3 Sources of Energy Waste

Following on from the preceding section, we now identify the following four different factors that negatively impact the energy efficiency of a streaming service:

1. *Downloading content at a rate lower than the maximum possible rate*: The most energy efficient way to transmit data at a given moment is to use the full capacity of the wireless channel. This holds for all the types of wireless access (Wi-Fi, 3G, LTE). The reason is that the lower the rate, the greater is the time interval between the packets. Even though these intervals are small, the inactivity timers are active and the smartphone consumes energy.

2. *Auxiliary control traffic during periods when content is not being received*: As we observed with the ON-OFF strategy using a persistent TCP connection, this kind of traffic has a major impact on the energy efficiency because it keeps the client's radio powered on all the time in the worst case.

3. *Tail energy in non-continuous content reception*: The non-continuous download strategy, such as the ON-OFF-M, causes tail energy to be spent after the download of each chunk of content. This energy adds up to a significant amount especially when using 3G or LTE and if chunks are relatively small.

4. *Downloading unnecessary content*: In cases where users always watch the whole video, the most energy efficient streaming strategy would be Fast Caching because it

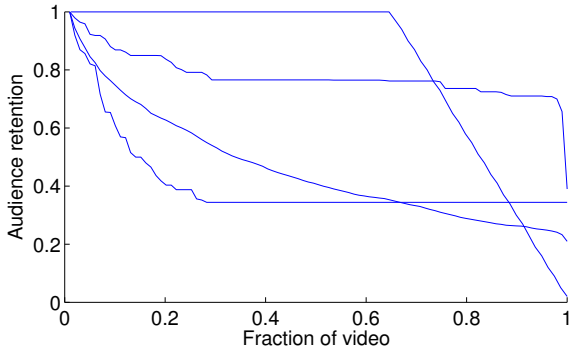


Figure 3: Audience retention graphs from YouTube videos.

minimizes the amount of time that the radio is kept powered on. However, since users often interrupt their viewing before the end of the video clip, such aggressive prefetching leads to unnecessary consumption of energy to download content that the user will never watch.

We note that while the first and second problems can be solved, for example, by alleviating server-side rate limits and by using non-persistent connections, the third and fourth problems are in fact contradictory: By downloading larger chunks to reduce the total amount of tail energy, the probability of downloading unnecessary content increases. We examine this tradeoff in the next section and propose an algorithm which optimally balances this tradeoff.

3. VIDEO STREAMING BASED ON CROWD-SOURCED VIEWING STATISTICS

3.1 Audience Retention

The fact that users do not always watch entire video clip transforms finding an energy optimal download schedule from using the trivial Fast Caching strategy into an optimization problem. The solution must strike a balance between two sources of energy waste: 1) prefetching content in large chunks in order to minimize the tail energy and 2) limiting that chunk size in order to reduce the amount of downloaded content that will never be viewed.

A key question in devising a solution is to estimate at what point a user will abandon watching the video. We propose to leverage crowd-sourced video viewing statistics. The underlying idea is that video clip specific statistics collected from all the previous viewing sessions give useful insights about how a new user will watch the video. It is natural that the viewing behaviour depends on how interesting and "capturing" or engaging the video clip is and the content type. We study the potential of using such statistics in download scheduling to save energy. We show that it helps to save significant amount of energy compared to strategies that do not leverage crowd-sourced statistics.

One source of such statistics is YouTube, which maintains *audience retention* data for each video clip [6]. These statistics are computed based on continuous feedback sent by the YouTube client applications during video streaming sessions [23]. Figure 3 shows examples of audience retention curves for some videos which we extracted from YouTube. Each curve always starts from one and shows, for a particular video clip, the fraction of the audience that is still watching at a given point of time. These examples highlight the fact that retention highly depends on the video clips: some retain clear majority of audience until the end while others lose most of the audience at the very beginning of the video.

3.2 eSchedule Algorithm

We designed a download scheduler that takes viewing statistics, such as the audience retention plotted in Figure 3, as input. We name it eSchedule. We assume that a mobile device is able to tell the scheduler at each moment the type of the network that it uses for data communication. In addition, it includes parameterized power models of communication using each different technology (i.e., Wi-Fi, 3G, and LTE including variants with and without DRX). Using the interrupt probabilities computed from the crowed-sourced viewing statistics, the scheduler computes the optimal download schedule, which is a sequence of variable size chunks that is estimated to minimize the energy consumption. Intuitively, if the amount of tail energy is large, e.g. when using 3G, and the viewing statistics suggest that many users watched a large fraction of the video, the scheduler will choose to download the large chunk of content. Conversely, if the amount of tail energy is small, e.g. when using Wi-Fi, and the statistics indicate that audience retention has been poor, the scheduler chooses small chunk sizes. Next, we model this approach as a stochastic optimization problem.

We divide the video viewing time into n discrete time steps. Then, we consider the video viewing as a discrete time stochastic process $p_i(X)$, where X takes value one or zero and describes whether the user watches the video segment corresponding to the ith time step. The audience retention A_i (also defined in discrete time steps) describes the joint probability of watching all the segments from the beginning till the ith time step, i.e. $A_i = \prod_{k=1}^{i} p_k(X = 1)$.

The algorithm computes for each schedule an expected value of energy waste. We define the energy waste as the total tail energy and energy spent downloading data which is not watched. We obtain (1) for calculating the expected value of energy waste when downloading a video chunk corresponding to T seconds of content starting from discrete time step i. In this case, T represents an integer value. In the equation, btc is TCP bulk transfer capacity, P_{rx} is receive power, $E_i^{tail}(T)$ is the amount of tail energy for a chunk carrying T seconds worth of content, and $\mathbb{E}[B_i^{waste}(T)]$ is the expected value of content that is downloaded and never watched because the user interrupts the session.

$$\mathbb{E}[E_i^{waste}(T)] = E_i^{tail}(T) + \frac{\mathbb{E}[B_i^{waste}(T)]}{btc} \times P_{rx} \qquad (1)$$

Computing the tail energy is straightforward given that the algorithm knows the power states and inactivity timer values for each specific radio interface. This tail energy depends on the chunk size because the inactivity timer values corresponding to a specific radio interface can exceed T. The tail energy is added for a chunk even if the user interrupts viewing in the middle of downloading it. When downloading a chunk containing T seconds worth of content, the expected value of the unnecessarily downloaded content is calculated using (2). This equation consists of two parts: the first one corresponds to the phase during which the chunk is being

simultaneously downloaded and consumed, and the second one to the phase during which it is only being consumed.

$$\mathbb{E}[B_i^{waste}(T)] = \sum_{k=1}^{\frac{T\times r_s}{btc}} p_i^{int}(i+k)\big[btc\times k - k\times r_s\big] + \sum_{k=\frac{T\times r_s}{btc}+1}^{T} p_i^{int}(i+k)\big[T\times r_s - k\times r_s\big] \quad (2)$$

In the above equation, r_s is the encoding rate of the streaming. We also need the probability of the user interrupting viewing (p_i^{int}). This probability needs to be re-evaluated each time the next chunk size is chosen by the scheduler during the streaming session. It is a conditional probability because it depends on the current viewing position and in addition on the audience retention. Therefore, it does not matter whether the viewing is continuous or not, since the probability to interrupt viewing is always computed from the current viewing position and it can be the new location in case of a forward or backward skip by the user. Thus, the probability for a user to interrupt during a discrete time step j evaluated at time step i (i.e., user is currently viewing at position i, $j > i$) is computed as follows:

$$p_i^{int}(j) = p(\text{“abandon at j”}|\text{“curr. position i”}) = \frac{A_{j-1}p_j(X=0)}{A_i} = \frac{A_{j-1}(1-p_j(X=1))}{A_i} = \frac{A_{j-1}-A_j}{A_i}$$

Each chunk can be of any size ranging from a defined minimum to the remaining video duration. Consequently, the number of all possible schedules grows exponentially as the video length grows. Even if we limit ourselves to considering only a certain granularity of chunk sizes, e.g. multiples of five second, it is infeasible to compute the number of all possible schedules, even for a video lasting only a few minutes (e.g., one minute video has more than two million possible schedules). The problem can be viewed as shortest path computation in which different sequential chunks represent different cost hops. The most efficient known algorithms for computing the shortest path scales according to $O(|E|+|N|log|N|)$, where $|E|$ is the number of edges and $|N|$ is the number of nodes. Our scheduling problem requires a path corresponding to each possible schedule which means that the number of edges grows exponentially as the video length extends, and it is infeasible to compute an optimal solution using exhaustive search. Hence, we use a heuristic, described in the next section.

3.3 Heuristic

Our approach is to limit the number of consecutive chunks when choosing the optimal size for the subsequent chunks. In this way, the algorithm finds a locally optimal schedule, i.e. it minimizes energy waste for the next chunks to be considered, but finding a globally optimal schedule is not guaranteed.

The heuristic we use is the expected value of energy waste per content download in Joules per second. The expected value is again calculated as shown in (1) and we obtain (3) as the heuristic.

$$Hr = \sum_{k=1}^{|S|}(\mathbb{E}[E_i^{waste}(T_k)])/\sum_{k=1}^{|S|}T_k \quad (3)$$

eSchedule searches for a variable size chunk download schedule $S = (T_1, T_2, T_3, \ldots, T_n)$. Each time a new video chunk download needs to be scheduled, i.e. every time the amount of content in the playback buffer goes below a minimum threshold, the algorithm finds such an S that minimizes Hr and then selects T_1 as the next chunk size. The size of S limits the number of consecutive chunks to consider in each scheduling decision. According to our tests, which used a video lasting ten minutes, having more than two consecutive chunks in the scheduling yields only minor improvements in overall energy efficiency.

4. PERFORMANCE EVALUATION

4.1 Simulation Setup

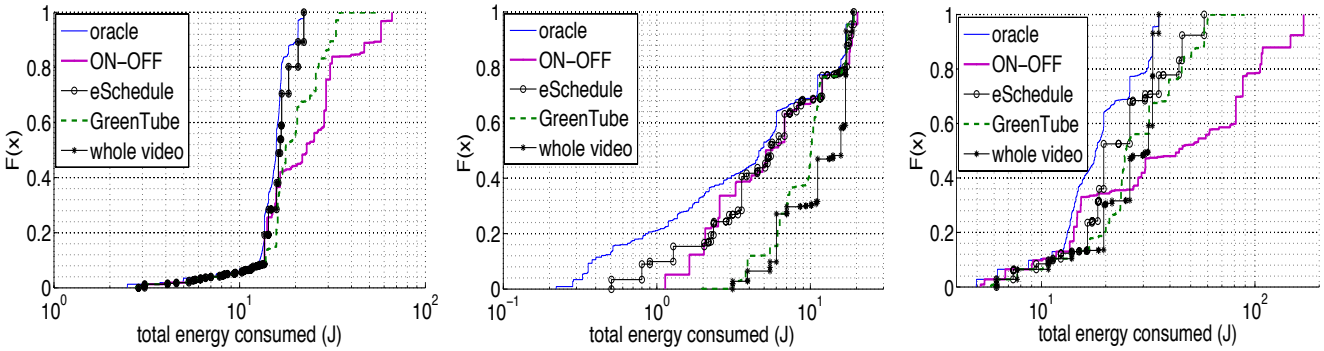
We implemented eSchedule in Matlab in order to compare it to the existing alternative approaches. Our colleagues helped us to collect audience retention information from 16 different YouTube videos[1]. Their durations range from one to eleven minutes and they had been viewed from a few hundred to five million times. Average stream encoding rates vary from 160 Kbps to 800 Kbps. The content includes recordings of illustrated discussions, music performances, advertisements, and martial arts exhibitions and lessons.

We divided the set of videos into short and long ones. The 10 short ones last less than five minutes and the 6 long ones more than five minutes. We simulated 1000 video streaming sessions for one user separately for both sets of videos. We simulate the behaviour of an average user: For each session, the video is randomly selected from the set and the abandoning time of viewing is randomly drawn from a probability distribution following the audience retention information of the selected video. For example, if the audience retention of a given video tells that not a single user abandoned the viewing during the first ten seconds, the simulations would also never draw an abandoning time shorter than ten seconds for that video. The video download schedule was computed with each considered algorithm, and the total energy consumed was calculated for each session.

We compare eSchedule to four other approaches: 1) to Oracle, which knows the viewing time for each session and downloads the content corresponding to that duration in one shot, 2) to downloading whole video at once, 3) to ON-OFF-M downloading 60-second chunks of content, which roughly corresponds to default YouTube Android app, and 4) to the algorithm used by GreenTube [16].

GreenTube resizes the download buffer size based on the user's viewing history stored locally in smartphones. It first computes an expected value of viewing time using this history. In turn, it finds a buffer size which will minimize the total energy consumed between the current and the end of the expected viewing time, also taking into account the power consumed during download and the tail energy. We implemented this algorithm, including maintaining local viewing history of the simulated user, on Matlab as well. In this way,

[1]These statistics are currently available only for the video owner in YouTube.



(a) short video over LTE (b) long video over Wi-Fi (c) long video over LTE

Figure 4: CDF plots of the downloading energy consumption when streaming videos.

WNI	tail timers	power values	datarate
Wi-Fi	PSM: 0.2s	P_{rx} = 0.78W, P_{tail} = 0.435W	10Mbit/s
3G	FD: 5s	P_{DCH} = 0.78W, P_{tail} = 0.78W	6Mbit/s
LTE	inactivity: 10s	P_{rx} = 1.58W, P_{tail} = 1.3W	16Mbit/s
LTE+DRX	DRX: 0.75s	P_{rx} = 1.58W, P_{tail} = 1.3W	16Mbit/s

Table 1: WNI timer configurations and power consumption at different states of the interface.

we are able to compare our approach of using crowd-sourced per-video viewing statistics to GreenTube's approach of using local viewing statistics.

At the beginning of simulations, GreenTube's local viewing statistics were initialized to null as we considered that to be as realistic as initializing it with random statistics. They are updated according to the simulated viewing behaviour throughout the 1000 viewing sessions. In addition, the lower threshold of buffer size was reduced to 1 MB in order to allow GreenTube to use also small buffer sizes and to make the comparison fair.

Table 1 summarizes the network and power related parameters used in simulations. The data rates imply end-to-end bulk transfer capacity rather than the over-the-air data rate of wireless access network technologies. Most of the other parameters were identified during the measurements presented in Section 2.2. Only the DRX timer was assumed according to a vendor recommendation as the DRX was not enabled in the commercial network. The power models used are straightforward in that we assume a fixed power draw during each different state, namely reception of data, idle but radio on (tail power), and sleep. We used power values relative to each idle state of each WNI, namely 3G's CELL_PCH, LTE's idle, LTE's DRX sleep, and Wi-Fi's PSM sleep modes. Note that for 3G access we assumed Fast Dormancy as the Galaxy SIII supports. In the case of LTE, we considered both cases; with and without DRX.

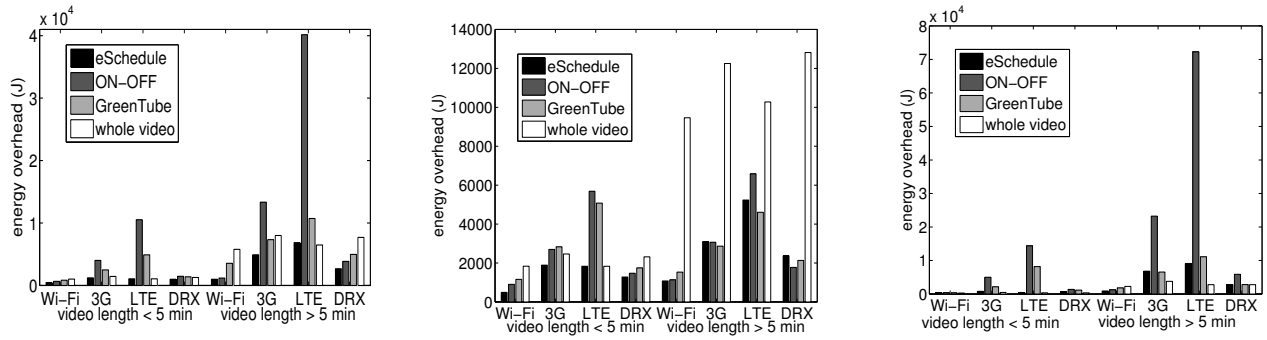
4.2 Energy Consumption

In the first simulation round, we only included videos that are shorter than five minutes. The results show that downloading the entire video in one go is energy efficient in the case of short videos, especially when the amount of tail energy is high, such as with LTE (Figure 4(a)). Similarly, eSchedule responds to short videos by typically choosing in almost every case to download the whole video in one go. By contrast, for those videos longer than five minutes, we observe that downloading the whole video is less attractive. The performance of the ON-OFF strategy is fairly good when the tail energy is small, such as in the case of Wi-Fi (Figure 4(b)). However, when streaming over 3G and LTE, which exhibit large tail energy, the results are significantly worse (Figure 4(c)).

We computed the energy consumption overhead by subtracting the energy consumed by Oracle from the energy consumed by other strategies for each streaming session. We sum up this overhead for each test case and plot the results in Figure 5(a). In every configuration, eSchedule achieves overall the lowest energy consumption. On average, *eSchedule cuts the energy waste to approximately half when compared to the next best approach.*

The behaviours of the other competing download strategies are intuitive but GreenTube deserves more analysis. The reason why GreenTube in most cases consumes more energy than eSchedule is two fold. First, unlike eSchedule, which leverages "global" video viewing statistics, GreenTube uses "local" statistics and assumes that a given user will always watch each video in a similar fashion regardless of the content, thus, for example, the user will either always watch just a small fraction of each video or always watch the entire video. The vastly different shapes of audience retention graphs undermine this assumption.

The second reason is related to the way GreenTube optimizes its buffer size. To illustrate the problem, let us consider two situations: In the first one, the user has always watched roughly half of the video, and in the second, the viewing history is evenly distributed. Both cases suggest that the expected viewing time is half the video and thus GreenTube will choose the same download strategy. However, in the latter case, it is much less likely that the expected value will provide an accurate estimate. The consequence is large especially when the tail energy is significant, in which case an underestimated viewing time yields a large penalty (note the logarithmic scale in Figures). By way of comparison, eSchedule computes the download schedule based on the expected value of energy wasted and, thus, also considers the "shape" of the statistics when deciding the next chunk size.



(a) average user (b) impatient user (c) patient user

Figure 5: Total energy overhead (J) to watch 1000 videos for three kinds of user compared to Oracle.

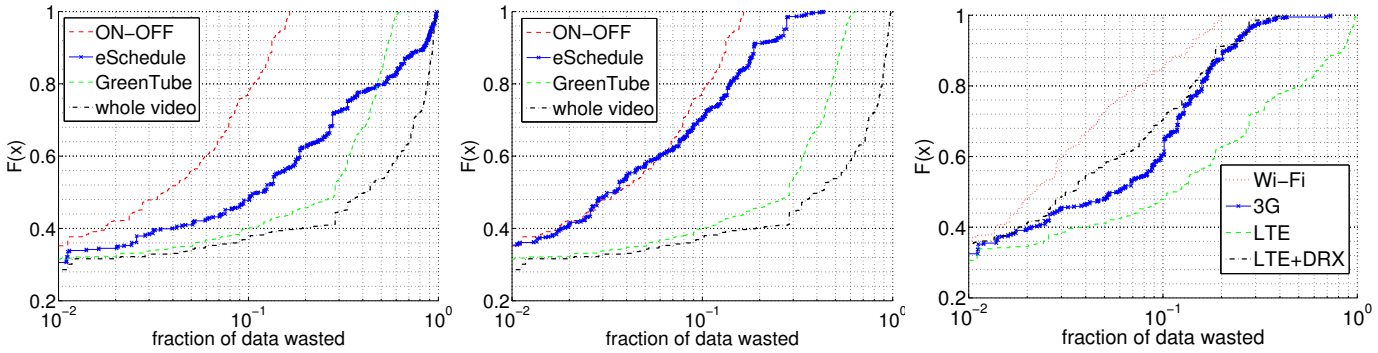


Figure 6: ON-OFF spends extra energy for less data waste when streaming long videos over LTE.

Figure 7: eSchedule and ON-OFF download least unnecessary data when streaming over Wi-Fi.

Figure 8: The smaller the tail energy, the less unnecessary content is downloaded when using eSchedule.

4.3 User Bias

To understand the resulting energy consumption when the user deviates from the average behaviour, we also simulated two different user types which we term *impatient* and *patient*. The impatient user always abandons watching a particular video before the median user did according to the audience retention, while the patient user always continues to watch at least as long as the median user did. We again randomly draw the viewing time from the probability distribution following the audience retention but this time only from the first or second part of it for the impatient and patient users, respectively, the boundary being the time where 50% of users had abandoned the session.

The overall results are shown in Figure 5(b) and 5(c). First thing to notice with respect to the "impatient" user behaviour is that downloading the whole video is a poor strategy if the video is relatively long. GreenTube does a better job with the long videos because the user behaviour is *consistently biased* in which case local viewing history helps. eSchedule performs roughly similarly with it when viewing long videos but better with the short videos. Users that continue viewing longer than the median user are most penalized by the default ON-OFF strategy because of the tail energy associated with each chunk. Downloading the whole video is a good strategy in this case. Again, eSchedule and GreenTube perform similarly with long videos but eSchedule does a better job with the short ones. We discuss in Section 6 how eSchedule can be extended to adapt to this kind of situations to provide even better performance.

4.4 Data Waste

A download schedule that minimizes energy consumption does not always minimize unnecessary content download. The reason is the tail energy. Indeed, the larger the tail energy, the more favorable an aggressive buffering will generally be from the energy consumption perspective. Yet, limiting the amount of unnecessarily downloaded data is important for some mobile network users who have a quota in their data plan. It is equally important, too, for ISPs, who need to provision networks based on traffic volumes.

We computed the amount of content downloaded but not watched for all the simulated scenarios. Figures 6 and 7 plot the data waste for the scenarios where videos longer than five minutes are streamed over LTE both without and with DRX. These figures illustrate the role of the amount of tail energy. The results demonstrate that downloading the whole video in one go leads to a lot of data waste: half of the sessions download roughly 40% of the content unnecessarily. Using GreenTube also leads to considerable amount of data waste: half of the sessions download approximately 30% of the content unnecessarily. Compared to these two, the data waste generated by eSchedule is on average much more reasonable: approximately 10% when streaming over LTE and only about 3% when DRX is enabled for half of the sessions. Figure 8 confirms that the amount of content unnecessar-

Algorithm 1 StreamThrottler

```
bytes = 0, brate = 0, bwidth = 0, faststart = 40;
contentlength = 0, videodownloaded = false
audret = get_audience_retention()
while videodownloaded == false do
    pos = get_playback_pos()
    if (bitrate == 0) then chunksize = 5
    else
        chunksize = eSchedule(radio, brate, bwidth, pos, audret)
        consdown = chunksize/faststart
    for i = 1 → consdown do
        shortchunk = calculatechunk(chunksize)
        request = formatrequest(bytes, contentlength)
        response=download(shortchunk,bitrate,request)
        bytes+ = process(response)
        if (brate == 0) then
            contentlength = process(response)
            brate = process(response)
            bwidth = process(response)
    if (bytes == contentlength) then
        videodownloaded = true
     sleep(chunksize − 5)
```

ily downloaded by eSchedule depends on the size of the tail energy: the smaller it is, the smaller is also the expected data waste. It should be noted that neither GreenTube nor eSchedule were configured to specifically limit data waste in these simulations. In both solutions, it is possible to further limit the data waste by spending extra energy. In case of eSchedule, this can be achieved by specifying an upper limit for the estimated amount of unnecessarily downloaded content, i.e. the algorithm would reject all schedules where $\mathbb{E}[B_i^{waste}(T)]$ exceeds that limit in (2).

5. StreamThrottler: Android App

In order to show that eSchedule works also in practice, we designed and implemented a YouTube client for Android and experimented with it. This client comprises two components: the eSchedule and a downloader. We next describe both of these in turn before presenting the results obtained from testing the application.

5.1 eSchedule Implementation

We implemented eSchedule according to the description in Section 3. Since audience retention information is currently available only to the video owner, we used previously collected data. In addition, the algorithm requires as inputs the encoding rate of the stream, the download rate, and the wireless interface being used and their respective power models. The downloader finds the encoding rate by parsing the first few hundred bytes of the video header. It also measures the rates and determines the interface being used.

eSchedule also takes the current position of playback as input. If the user does not watch the stream continuously but instead jumps forward or backward while streaming, eSchedule simply computes the next chunk size taking into account the audience retention data when starting from the new position.

We implemented power consumption models for Wi-Fi, 3G, and LTE in the same way as we did for the Matlab evaluation (Section 3) because there are no open APIs in a mobile device to obtain information about the states of the 3G/LTE RRC protocols. We implemented the 3G power model knowing that both the testing device, Galaxy SIII, and our commercial HSPA network support Fast Dormancy having a 5 second inactivity timer. In the case of LTE, we also used a commercial network and found the RRC inactivity timer to be 10 seconds, which is commonly used.

5.2 Downloader

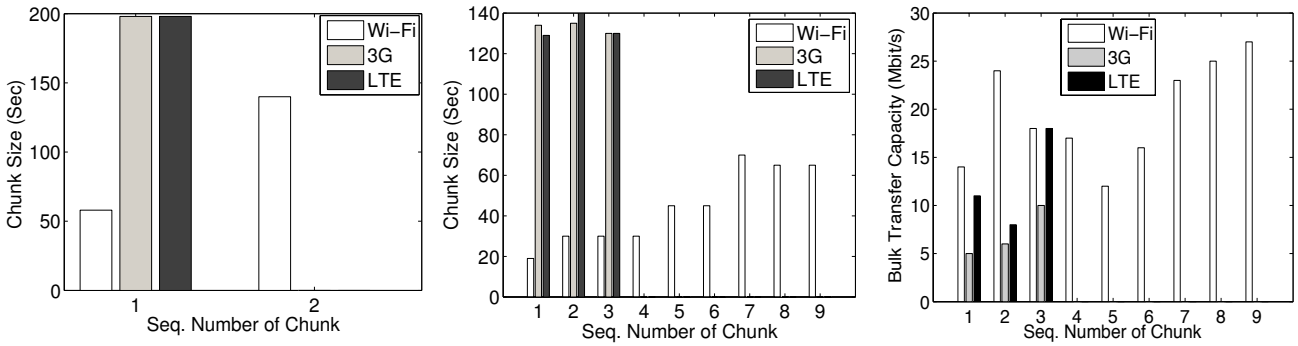
In order to start a new streaming session, StreamThrottler's downloader retrieves the YouTube HTTP video download URL and the duration of the video from the server, in the same way as a default player does. It then downloads video content from the server in chunks whose sizes are determined by the eSchedule.

The download rate and stream encoding rate are typically unknown beforehand. Therefore, the downloader uses an initial chunk of 5 seconds during which it measures the encoding and download rates and, in turn, lets eSchedule make the first scheduling decision. If the decision is to download more than 5 seconds worth of video as the first chunk, the downloader makes immediately another HTTP request and continues downloading the content after the first five seconds chunk has completely arrived. We use five seconds as the minimum chunk size.

Our traffic and power measurements in Section 2.2 suggest that the streaming services control the rate of download after the Fast Start phase, and that these methods lead to energy waste for the streaming clients. We want StreamThrottler to avoid this problem in a case where eSchedule decides to download a chunk of a size that exceeds the amount downloaded during the Fast Start phase. Therefore, we apply a maneuver to bypass the server's rate control mechanism by exploiting Fast Start. What happens is that a single TCP connection is used to download only the amount of content that the server delivers in the Fast Start phase, during which no rate limit is applied. After that phase, the connection is closed and another one is established. A new HTTP request for the subsequent range of content is sent over the new TCP connection and again the amount of content equivalent to the Fast Start phase will be downloaded without a rate limit. This behaviour continues iteratively until the amount of content equivalent to the complete chunk has been downloaded. In this way the downloader can download content at the maximum end-to-end bandwidth and can reduce the active usage time of wireless interfaces significantly. The pseudo code of the downloader is shown in Algorithm 1.

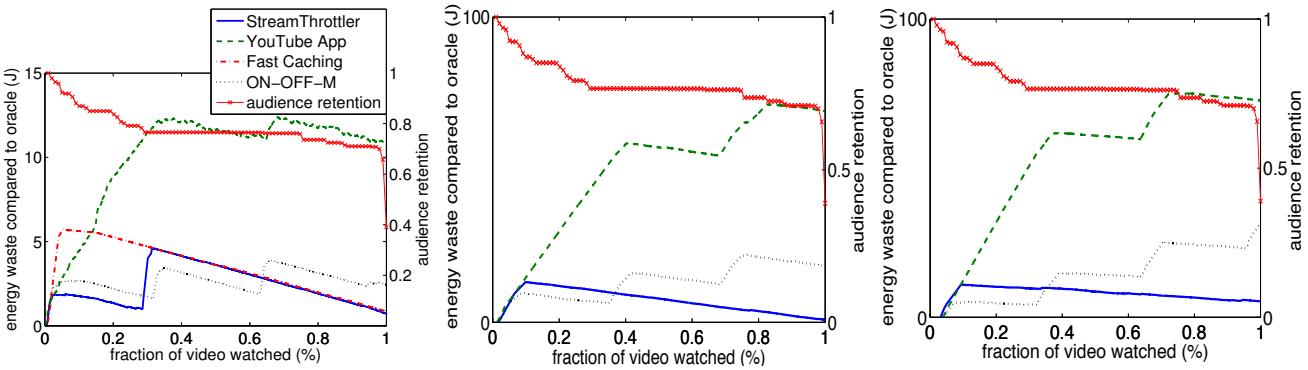
5.3 Dealing with Wireless Latency

eSchedule may decide to download a video in multiple chunks either when the WNI has small tail energy or when the audience retention shows little probability of viewing the whole content. Downloading content in chunks may increase the chance of letting the playback buffer run dry in between chunks. This probability is higher for a smartphone compared to a device connected to the fixed network because the mobile device spends some time accessing the wireless channel before beginning to download a chunk. This latency is not significant for Wi-Fi and LTE access. However, when using 3G access such a delay may be up to 3 seconds because of transitioning from IDLE or CELL_PCH to CELL_DCH state [19]. Therefore, it is practical to let the client to download the next chunk before the playback buffer is completely empty. StreamThrottler also does this by downloading the new chunk 5 seconds in advance. We considered this five second value based on the analysis present in [27].



(a) 198 s (b) 399 s (c) Bulk transfer capacity (399 s)

Figure 9: Chunk sizes in seconds chosen by eSchedule and the downloading rates calculated by the downloader.



(a) Wi-Fi (b) 3G (c) LTE

Figure 10: Estimated energy waste compared to Oracle while streaming the 198s video using Galaxy SIII.

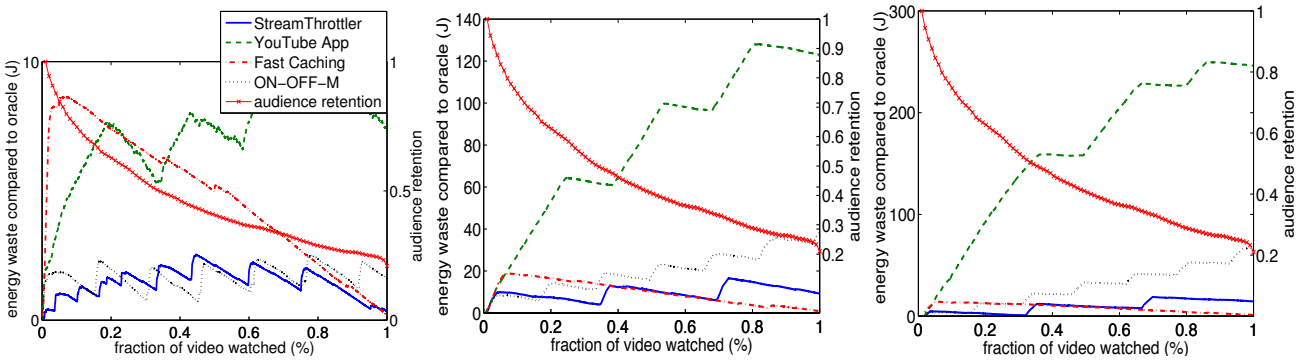
5.4 Testing and measurements

In this setup, the Wi-Fi access point was connected to the 100 Mbit/s university network and the smartphone had unlimited access to the cellular networks. We streamed two YouTube videos, a short (198s) and a long (399s) one, on Samsung Galaxy SIII 4G using (i) StreamThrottler, (ii) the default YouTube app, (iii) Fast Caching and (iv) an ON-OFF-M download strategy without throttling which downloads 65 s equivalent chunk after every 60 seconds. Both videos are of 360*p* quality. We compare these four test cases by measuring the current drawn by the smartphone using the Monsoon Power Monitor [1]. Again, as we did in Section 2.2, we subtracted the manually identified playback current from the total current consumption because of different apps and in order to compare only the current drawn by downloading.

Figure 9 shows the chunk sizes and downloading rates determined by the StreamThrottler. To illustrate the key differences in the download strategies and mechanisms, we compute the energy waste for each streaming session as a function of time. This waste is computed by subtracting the energy consumption estimate calculated for Oracle as a function of time from the cumulative energy consumption measured for above mentioned four techniques. The Oracle strategy downloads the right amount of content in each case. We assume that Oracle would on the average draw the same amount of current while downloading content and achieve the same average download rate than StreamThrottler.

The results for the short video and the corresponding audience retention for that video are plotted in Figure 10. The corresponding chunk sizes chosen by eSchedule are illustrated in Figure 9(a). What stands out first is the striking difference in overall cumulative energy consumption between StreamThrottler and the default app. This difference is mainly due to the low download rate resulting from server throttling and tail energy, in turn, due to multiple chunks (see Section 2.3), especially when streaming over 3G and LTE. Therefore, the difference becomes smaller when ON-OFF-M without throttling is used.

The second noticeable feature is that StreamThrottler, which uses eSchedule, adapts the download strategy to the audience retention and the characteristics of wireless network. When streaming over 3G and LTE, eSchedule chooses to download the whole video in one shot. The reason is the large amount of tail energy and relatively strong audience retention. Therefore, ON-OFF-M (without throttling) causes more energy waste than StreamThrottler resulting from more tail energy. As for streaming over Wi-Fi, eSchedule chooses to split the download into two chunks because the penalty arising from tail energy is clearly smaller. A careful observation reveals that the chunk sizes are chosen in such a manner that the first one is sufficient to cover



(a) Wi-Fi (b) 3G (c) LTE

Figure 11: Estimated energy waste compared to Oracle while streaming the 399s video using Galaxy SIII 4G.

viewing until the end of the first drop in the audience retention (first 30% of the video). The second chunk then covers the rest of the video during which almost no viewer interrupted their viewing – this is is visible as an almost flat retention curve. In this way, eSchedule does not risk a hefty energy waste caused by potential abandonment during the first 30%.

Similar results for the long video are plotted in Figure 11 and the corresponding chunk sizes eSchedule chose are shown in Figure 9(b). The viewing statistics suggest that users are continuously abandoning the viewing but the rate of abandonment is decreasing as the viewing progresses. For this reason, when streaming over Wi-Fi, eSchedule chooses chunks that are of increasing size while the viewing progresses. The much larger tail energy of 3G and LTE causes eSchedule to choose larger chunk sizes, as occurs with the short video, but this time the video is split into three chunks because the probability of a user abandoning the video is much higher than in the case of the short video. On the contrary, ON-OFF-M uses more chunks and causes more energy waste using 3G and LTE. At a first glance, it would seem that Fast Caching performs better. However, from the audience retention graph we can see that the video loses around 50% of its viewers during the first 35% of total duration and during that time there is more energy waste than the StreamThrottler.

We also measured the computational overhead of eSchedule. To do that, we measured the current consumption of the phone running StreamThrottler and computed the download schedules using eSchedule against those selecting chunk sizes from hardcoded values. No actual downloading took place. We chose to use the long video over Wi-Fi as the test case because that requires more frequent scheduling decisions than the other ones. The measured average extra power draw caused by eSchedule computation was less than 15 mW which is negligible compared to the total power draw including playback and download power.

6. PRACTICALITIES

A relevant question concerning the sources of energy inefficiency is why video service providers do not use encoding rate streaming or throttle the transmission rate. The answer cannot be network congestion because that is not an application level problem and it is handled by TCP. One conclusion might be the concern about "bandwidth waste", i.e. the bandwidth used to deliver content to users which is never consumed because of abandoning the session before the end. eSchedule would eliminate such concern because this download scheduler uses users' viewing statistics. Furthermore, eSchedule could be tuned to emphasize the importance of the bandwidth waste by setting an upper limit to a chunk size, as we discussed in Section 4.

Chunk size also can be limited when the device does not have enough memory to accommodate the content being downloaded. In that case, the device must spend extra energy. Although, a significant amount of YouTube videos are less than 10 minutes long [12], the size a 10 minutes long HD video (2 Mbps) can be 150 MB. However, modern smartphones have few hundred megabytes to two gigabytes of phone memory. If this memory is full then content can be downloaded to the SD card.

The design of eSchedule assumes that the amount of available bandwidth is greater than the encoding rate. However, the inverse is possible. If such situation is of persistent type, the content cannot be downloaded in time and the session is interrupted and eventually abandoned. In that case, any download schedule keeps the radio continuously on because a chunk download cannot finish before the next one should already be started. Dynamic Adaptive Streaming over HTTP (DASH) [24] is a good solution to cope with transient bandwidth fluctuations. This is because DASH allows to switch to a lower bit rate upon bandwidth degradation and it is possible to integrate eSchedule to a DASH client. In case of quality switch, a new chunk size calculation is triggered.

Concerning the audience retention information, we assume that other video services also collect crowd-sourced video viewing statistics in a similar manner to YouTube, but this data might be available only to the owner of the video in question. We can think of two ways to make such data available to eSchedule in the future. First, the video service provider opens up access to this data through an API. Should this data be deemed sensitive, however, the other option is for the video services to implement eSchedule in their own players, which then requests the viewing statistics from the server using some secure API only when playing a video. Yet another solution is to let the servers to do the scheduling.

In case a user behaves in a consistently biased way, at least for some time, it is possible to make eSchedule adapt to it. We can maintain a measure of user bias with respect to the average user by comparing the expected viewing time of a video computed from the audience retention data to the actual viewing time. This local bias measure could be used to adjust the interrupt probabilities when calculating the download schedule.

7. RELATED WORK

Energy consumption for multimedia streaming in mobile devices has been studied for the last decade. Most earlier studies have focused on streaming over Wi-Fi. A few recent papers consider energy consumption while streaming over cellular access network. The proposed solutions work from different vantage points and on different layers of the Internet protocol stack. In the following, we discuss the most important parts of that body of work and compare it to our work in this paper.

Anand et al. proposed adapting Wi-Fi power management in response to access patterns and application hints on data access [7]. Bagchi et al. [9] also presented a solution for streaming over Wi-Fi, one in which a player maintains a fixed playback buffer size and requests data from the server when the buffer status touches a low buffer watermark. The alternative is that the player plays from the playback buffer and ceases downloading. This solution is similar to the ON-OFF-S mechanism discussed in Section 2. Two other similar approaches [28, 25] modify TCP window-size of IP packets to trigger the TCP flow control mechanism artificially, which eventually generates bursty traffic when the TCP window-size is modified to a higher value. These solutions save energy by forcing the Wi-Fi into sleep mode, which means they avoid any potential auxiliary traffic such as TCP control packets, and, at the same time, the application layer keeps messages alive. However, these solutions do not address the problem of users abandoning the viewing of the video.

Several kinds of optimization techniques exist to reduce energy while communicating over 3G. For example, RadioJockey mines program execution in order to optimize the use of Fast Dormancy in 3G communication and in that way reduces energy consumption [8]. However, it does not actually target streaming; instead it focuses on background apps. ARO [18] and TOP [20] have similar objectives but they require active participation from application developers. Bartendr saves energy by prefetching streaming content when the signal strength is good [22]. It is in fact a complementary mechanism that could be integrated into eSchedule.

Some work has looked at scheduling of mobile traffic based on the power consumption characteristics of wireless radios. For example, CoolSpots propose to schedule low bit rate video transmission over Bluetooth and higher bit rate video over Wi-Fi [17]. Such a solution is orthogonal to what we propose in this paper. If eSchedule is used with a downloader that fetches each chunk of video over a separate TCP connection, such as StreamThrottler, CoolSpots could complement the system to save more energy by downloading chunks over Bluetooth when deemed beneficial.

There are also many papers studying measurement and characterization that touch on the subject of our work. For instance, 3G energy consumption was studied in [10] and LTE power characteristics in [15]. Trestian et al. specifically studied streaming and energy consumption in [26].

The closest one to our work is GreenTube [16] was recently proposed as a solution to optimize energy consumption while streaming YouTube videos to Android phones. The download scheduling part of GreenTube has two important differences to eSchedule: First, it is based on users' local viewing history rather than global per-video statistics that we propose to use in this work. Second, the use of expected value of viewing time as the sole input to scheduler may lead to a significant energy waste if tail energy is large (see Section 3).

8. CONCLUSION

We highlighted a number of sources of energy inefficiency in wireless video streaming two of which are contradictory, namely tail energy spent because of downloading content in chunks and energy spent in prefetching content that the user will never watch. Consequently, we designed a novel download scheduling algorithm, eSchedule, which uses crowd-sourced video viewing statistics and wireless communication power models to predict user behaviour and optimize the tradeoff between these two sources of energy waste. Evaluation results show that eSchedule cuts the energy waste to half compared to the next best alternatives. We also implemented an Android prototype called StreamThrottler which implements eSchedule and a chunk-based YouTube downloader that speeds up the downloads by exploiting Fast Start technique. Experiment results confirm that the prototype exhibits the desired properties of eSchedule and, furthermore, saves up to 80% of overall energy.

Our future work includes integrating eSchedule into DASH and considering other additional input for the scheduler, such as signal strength indicators and events indicating network activity of other apps in order to leverage the opportunity of simultaneous transfers to further amortize tail energy. Another extension of this work is to take into account the user's local viewing history, in addition to the crowd-sourced viewing statistics per video as described earlier. In this way, we could add a personalization component to the download scheduling in order to further enhance the prediction of user behaviour.

9. ACKNOWLEDGEMENTS

This work was supported by the Academy of Finland: grant number 253860 and FIGS. We would also like to thank the anonymous reviewers, and our shepherd, Dina Papagiannaki, for their feedback.

10. REFERENCES

[1] Power Monitor, www.msoon.com.
[2] 3GPP TS 25.331, Radio Resource Control (RRC); Protocol specification, May 1999.
[3] IEEE 802.11, Wireless LAN Medium Access Control and Physical Layer Specification, 1999.
[4] 3GPP TS 36.331, E-UTRA; Radio Resource Control (RRC) Protocol Specification, May 2008.
[5] Fast Dormancy best practices, GSM association, network efficiency task force, 2010.
[6] Audience retention,`http://support.google.com/youtube/bin/static.py?hl=en&topic=1715158&guide=1714169&page=guide.cs`.

[7] Anand, M., Nightingale, E. B., and Flinn, J. Self-tuning wireless network power management. In *Proceedings of the 9th annual international conference on Mobile computing and networking* (2003), MobiCom '03, ACM, pp. 176–189.

[8] Athivarapu, P. K., Bhagwan, R., Guha, S., Navda, V., Ramjee, R., Arora, D., Padmanabhan, V. N., and Varghese, G. RadioJockey: mining program execution to optimize cellular radio usage. In *Proceedings of the 18th annual international conference on Mobile computing and networking* (New York, NY, USA, 2012), Mobicom '12, ACM, pp. 101–112.

[9] Bagchi, S. A fuzzy algorithm for dynamically adaptive multimedia streaming. *ACM Trans. Multimedia Comput. Commun. Appl. 7* (2011), 11:1–11:26.

[10] Balasubramanian, N., Balasubramanian, A., and Venkataramani, A. Energy consumption in mobile phones: a measurement study and implications for network applications. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference* (New York, NY, USA, 2009), IMC '09, ACM, pp. 280–293.

[11] Finamore, A., Mellia, M., Munafò, M. M., Torres, R., and Rao, S. G. Youtube everywhere: impact of device and infrastructure synergies on user experience. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference* (New York, NY, USA, 2011), IMC '11, ACM, pp. 345–360.

[12] Gill, P., Arlitt, M., Li, Z., and Mahanti, A. Youtube traffic characterization: a view from the edge. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement* (New York, NY, USA, 2007), IMC '07, ACM, pp. 15–28.

[13] Guo, L., Tan, E., Chen, S., Xiao, Z., Spatscheck, O., and Zhang, X. Delving into internet streaming media delivery: a quality and resource utilization perspective. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement* (New York, NY, USA, 2006), IMC '06, ACM, pp. 217–230.

[14] Hoque, M. A., Siekkinen, M., and Nurminen, J. K. Dissecting mobile video services: An energy consumption perspective. In *14th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks(WoWMoM)* (2013), WoWMoM '13, IEEE.

[15] 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. In *Proceedings of the 10th international conference on Mobile systems, applications, and services* (New York, NY, USA, 2012), MobiSys '12, ACM, pp. 225–238.

[16] Li, X., Dong, M., Ma, Z., and Fernandes, F. GreenTube: Power Optimization for Mobile Video Streaming via Dynamic Cache Management. In *Proceedings of the ACM Multimedia* (New York, NY, USA, 2012), acmmm'12, ACM.

[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 MobiSys 2006* (New York, NY, USA, 2006), ACM, 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 *Proceedings of the 9th international conference on Mobile systems, applications, and services* (New York, NY, USA, 2011), MobiSys '11, ACM, pp. 321–334.

[19] Qian, F., Wang, Z., Gerber, A., Mao, Z. M., Sen, S., and Spatscheck, O. Characterizing radio resource allocation for 3G networks. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement* (New York, NY, USA, 2010), IMC '10, ACM, pp. 137–150.

[20] Qian, F., Wang, Z., Gerber, A., Mao, Z. M., Sen, S., and Spatscheck, O. TOP: Tail Optimization Protocol For Cellular Radio Resource Allocation. In *Proceedings of the The 18th IEEE International Conference on Network Protocols* (Washington, DC, USA, 2010), ICNP '10, IEEE Computer Society, pp. 285–294.

[21] Rao, A., Legout, A., Lim, Y.-s., Towsley, D., Barakat, C., and Dabbous, W. Network characteristics of video streaming traffic. In *Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies* (New York, NY, USA, 2011), CoNEXT '11, ACM, pp. 25:1–25:12.

[22] Schulman, A., Navda, V., Ramjee, R., Spring, N., Deshpande, P., Grunewald, C., Jain, K., and Padmanabhan, V. N. Bartendr: a practical approach to energy-aware cellular data scheduling. In *Proceedings of the sixteenth annual international conference on Mobile computing and networking* (New York, NY, USA, 2010), MobiCom '10, ACM, pp. 85–96.

[23] Siekkinen, M., Hoque, M., K. Nurminen, J., and Aalto, M. Streaming over 3g and lte: How to save smartphone energy in radio access network-friendly way. In *Proceedings of the 5th ACM Workshop on Mobile Video* (2013), MoVid'13, ACM, pp. 13–18.

[24] Stockhammer, T. Dynamic adaptive streaming over HTTP –: standards and design principles. In *Proceedings of the second annual ACM conference on Multimedia systems* (New York, NY, USA, 2011), MMSys '11, ACM, pp. 133–144.

[25] Tan, E., Guo, L., Chen, S., and Zhang, X. PSM-Throttling: Minimizing energy consumption for bulk data communications in WLANs. In *Proceedings of International Conference on Network Protocols, 2007* (2007), IEEE, pp. 123–132.

[26] Trestian, R., Moldovan, A.-N., Ormond, O., and Muntean, G.-M. Energy consumption analysis of video streaming to android mobile devices. In *Network Operations and Management Symposium (NOMS), 2012 IEEE* (april 2012), pp. 444 –452.

[27] Wang, B., Kurose, J., Shenoy, P., and Towsley, D. Multimedia streaming via TCP: An analytic performance study. *ACM Trans. Multimedia Comput. Commun. Appl. 4*, 2 (May 2008), 16:1–16:22.

[28] Yan, H., Krishnan, R., Watterson, S. A., Lowenthal, D. K., Li, K., and Peterson, L. L. Client-centered energy and delay analysis for TCP downloads. In *IWQOS* (2004), IEEE, pp. 255–264.