A Client-Centric Data Streaming Technique for Smartphones: An Energy Evaluation

by

Abdulhakim Abogharaf

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2013

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

With advances in microelectronic and wireless communication technologies, smartphones have computer-like capabilities in terms of computing power and communication bandwidth. They allow users to use advanced applications that used to be run on computers only. Web browsing, email fetching, gaming, social networking, and multimedia streaming are examples of wide-spread smartphone applications. Unsurprisingly, network-related applications are dominant in the realm of smartphones. Users love to be connected while they are mobile. Streaming applications, as a part of network-related applications, are getting increasingly popular. Mobile TV, video on demand, and video sharing are some popular streaming services in the mobile world. Thus, the expected operational time of smartphones is rising rapidly.

On the other hand, the enormous growth of smartphone applications and services adds up to a significant increase in complexity in the context of computation and communication needs, and thus there is a growing demand for energy in smartphones. Unlike the exponential growth in computing and communication technologies, the growth in battery technologies is not keeping up with the rapidly growing energy demand of these devices. Therefore, the smartphone's utility has been severely constrained by its limited battery lifetime. It is very important to conserve the smartphone's battery power. Even though hardware components are the actual energy consumers, software applications utilize the hardware components through the operating system. Thus, by making smartphone applications energy-efficient, the battery lifetime can be extended. With this view, this work focuses on two main problems: *i*) developing an energy testing methodology for smartphone applications, and *ii*) evaluating the energy cost and designing an energy-friendly downloader for smartphone streaming applications.

The detailed contributions of this thesis are as follows: (*i*) it gives a generalized framework for energy performance testing and shows a detailed flowchart that application developers can easily follow to test their applications; (*ii*) it evaluates the energy cost of some popular streaming applications showing how the download strategy that an application developer adopts may adversely affect the energy savings; (*iii*) it develops a model of an energy-friendly downloader for streaming applications and studies the effects of the downloader's parameters regarding energy consumption; and finally, (*iv*) it gives a mathematical model for the proposed downloader and validates it by means of experiments.

Acknowledgements

All praise is due to Allah for granting me the strength and knowledge to complete this work. I would like to express my deep and sincere gratitude and appreciation to my supervisor *Dr. Sagar Naik* for his continuous encouragement, helpful discussion, and insightful guidance throughout the research presented in this thesis. I would also like to thank all my colleagues who created an excellent environment for the research work. Last but not least, I am indebted to my mother and brothers for their continuous support, patience, and prayers.

Dedication

To my parents.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

Mobile phones have become the favorite means of communication among users worldwide. According to the International Telecommunication Union (ITU), by early 2013, there are almost as many mobile users as people on earth [49]. PC-like smartphones have overtaken the ordinary mobile phones. That is because users appreciate portable devices combining the functions of cellphones with the functions of computers. By the end of 2011, about 16% of mobile subscribers in the world used smartphones with 25% annual growth, and about 51% of mobile users in developed countries used smartphones [48][47]. With this booming growth of smartphones worldwide, soon enough, smartphones will be more pervasive than ordinary mobile phones.

What actually helps smartphones to become so popular is the number of applications and services they provide. As of early 2013, there are about $800,000$ applications in each of the *Apple App Store* and the *Google Play Store* showing 60% growth from the year before. Users of different ages and interests enjoy a variety of mobile applications ranging from business to game applications, entertainment to educational applications and many more. The avalanche of progress in smartphone applications allows users to use their PC applications (similar to their PC applications) on the go. In actual fact, with the combinations of the built-in sensors in smartphones (such as accelerometers, gyroscopes, barometers, proximity detectors, light sensors, high-resolution cameras and microphones), smartphone applications are much more powerful than laptop applications. In addition to the context, the immediacy and simplicity are also advantages of smartphone applications.

The use of software applications as "computing" tools has become archaic. The age of software as a communication medium has arrived. Rather than the more traditional PC tasks of document creation and local computations, many of today's applications are communication-centered and based on data centers and cloud-computing. Users can use their computers indirectly through their smartphones. Network-related applications such as web browsing, email fetching, multimedia streaming and social networking are very popular mobile applications. It is expected that mobile phones will overtake PCs as the most common Web-access devices worldwide by the end of 2013 [17].

Streaming applications are gaining increasing popularity among other mobile network-related applications. Many TV businesses allow users to access live TV shows through their smartphones. Some other applications allow smartphone users to access media contents stored in their computers, and other applications enable users to watch live streaming contents broadcasted by other users. Not to mention, *Netflix*, *YouTube* and *Dailymotion* are among the most popular video-on-demand streaming websites. According to *Cisco*, mobile video traffic accounted for 51% of all mobile traffic by the end of 2012, and it is expected that two-thirds of the global mobile traffic will be video streaming by 2017 [28]. *YouTube* as the dominant application in mobile devices accounts for about 28% of all mobile traffic in North America [46].

Smartphones have been equipped with fast processors and lots of memory, which are comparable with those in their computer counterparts. Despite the limited size of handheld devices, it is easy today to find a smartphone with a 2 GB RAM and a 1.5 GHz dual-core processor, which were what laptops had just a few years ago. The significant evolution of the semiconductor industry, whose growth has been following *Moore's Law* of doubling the number of transistors in a chip every 18 months, makes it possible to have handheld devices with increasingly powerful processing capability, memory and screen resolution.

On the other side, the battery technology is not keeping pace with the growth of semiconductors. Whereas semiconductor technology is doubling every 18 months, the average annual gain in battery capacity is growing by only 5% [34]. The small size and light weight requirements of a handheld device imply that its battery be proportionately small in volume. Consequently, the system energy is severely limited. Table 1.1 shows the battery capacities of some smartphones released within the last few years. Clearly, in the last few years, there is not much growth in battery capacity compared to the growth in other smartphone resources.

Network related applications, more specifically streaming applications, are energy-hungry. It has been shown that streaming multimedia content consumes about 50% more energy than playing the multimedia content locally [8][60]. Streaming the content shortens

Phone	Battery Capacity	Phone	Battery Capacity
iPhone 3GS	1250 mAh	Samsung Galaxy S	1500 mAh
iPhone 4	1420 mAh	Samsung Galaxy S II	1650 mAh
iPhone 4S	1432 mAh	Samsung Galaxy Nexus	1850 mAh
iPhone 5	1440 mAh	Samsung Galaxy S III	2100 mAh

Table 1.1: Battery capacities of some recent smartphones.

the battery life time by half. Since multimedia streaming requires downloading the content involving the network interface usage, decoding the content involving the CPU usage, and playing the content involving the screen usage, it is not surprising that streaming applications are very energy-hungry. Since smartphone batteries are energy-poor, it is essential that mobile multimedia streaming applications avoid excessive energy consumption.

1.2 Problem Statement

The utility of smartphones is severely constrained by battery lifetime. So, it is important to reduce the amount of energy drawn from the battery by making the applications more energy efficient. In order to make smartphone applications more energy efficient, it is essential to have an effective and robust energy evaluation mechanism. Unlike the functional evaluation of an application, energy evaluation of an application is more challenging and requires more effort. Smartphones are highly configurable, and users can set the values of different parameters of the hardware components manually or via various utility programs attached to the operating system. For example, the volume parameter (which affects the energy consumption) of a device can be set to different values, or the Bluetooth parameter can be set to be discoverable. The states of the hardware components play a vital role in the energy consumption of smartphones, and it is important to consider the states of the hardware while evaluating the energy performance of an application. Due to the enormous number of phone configurations and application parameters, an immense number of test cases exists. Therefore, a selection technique reducing the number of test cases is desired.

Since streaming applications are energy-hungry, and since the energy cost of content delivery is high, evaluating the energy consumption of content delivery and designing an energy-friendly downloader for streaming applications get the priority. The energy consumption at the wireless network interface (WNIC) of a smartphone mainly depends on the answers to the following two questions: i) how much data is downloaded during the stream-

ing session? ii) how is such data downloaded? Downloading more data would make the WNIC work longer and consequently consume more energy. Practically, a user can choose a low quality version of the same multimedia content and expect less energy consumption. However, discovering how to download the content in order to reduce energy consumption is not that easy. Since all smartphones use WiFi interface cards which support the IEEE 802.11 power saving mode (PSM) by default, it is straightforward that the PSM should be utilized to minimize the energy consumption during streaming sessions. However, the standard PSM has limited effectiveness for streaming applications since streaming applications often involve bulk data transmission in a continuous fashion. Thus, the WNIC is not given many opportunities to switch into PSM.

The problems this thesis is trying to tackle can be best summarized in the following two points:

1. From a software development perspective, how is it possible to perform energy performance evaluation to smartphone applications?

2. Since streaming applications are energy-hungry, how is it possible to evaluate the energy consumption and design an energy-friendly downloader for mobile streaming applications?

1.3 Solution Strategies

The limited battery capacity of mobile devices presents an increasingly important challenge since data transmission consumes a lot of battery power. Energy efficient applications are prerequisite for saving energy in a mobile device. In order to evaluate the energy performance of an application, a methodology to select user-level test cases for energy cost evaluation of smartphone applications is needed. As shown in figure 1.1, smartphones interact with users and communication networks, and they draw battery energy based on the activities of the OS and applications running on top of the OS. The energy consumption of the device can be calculated by monitoring the power discharge rate from the battery. The energy consumed by an application can be estimated by taking the difference between the energy consumptions with and without running the application. When the application is not executed, the OS still consumes a certain amount of energy to make the device ready for running applications. In both cases, the settings of the smartphone parameters, termed as device configurations, play an important role even if an application does not access all the hardware components.

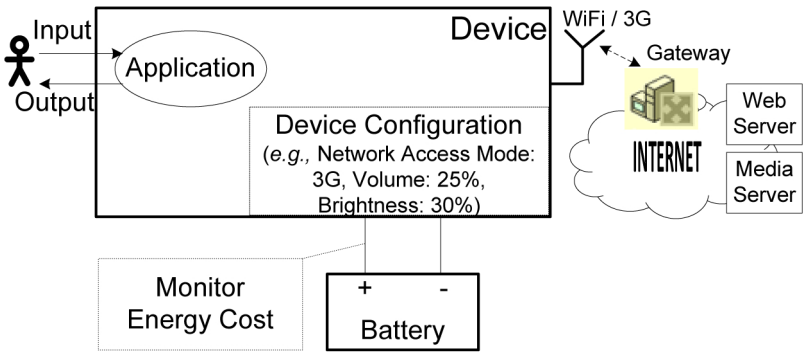


Figure 1.1: Energy performance test bench for smartphone applications

The work in [69] studied the parameters of five different state-of-the-art smartphones, and categorized them into three groups to reduce the number of test configurations for conducting energy performance testing. However, it dealt with a special case, where only one phone parameter is emphasized and considered in the *primary parameter* group. In this work, we generalize the methodology in [69] by allowing more than one parameter in the *primary parameter* group of the smartphone parameters. General expressions are derived to compute the number of test configurations for a given smartphone. We relate the work in [69] as a special case of this proposed work, and apply the methodology to *HTC Nexus One* smartphone as an example. We express the methodology in the form of a detailed flowchart that application developers can easily follow.

Since streaming applications are power-hungry, application developers should not only focus on the functionality aspect of their application, but also the energy performance aspect. Streaming applications consume energy at three stages of the streaming session: i) downloading the content; ii) decoding the content; and iii) playing the content. It is known that, for smartphones, the downloading cost is more significant than the decoding cost and the playing cost. Some application developers choose an energy-hungry downloading strategy without knowing the consequences. We evaluate the energy consumption of some popular streaming applications. We show that different streaming applications use different downloading strategies which lead them to consume different amounts of energy. We develop an energy-friendly streaming downloader and give a mathematical model of its energy cost. Then, we validate the mathematical model by means of experiments. We also study the effect of each of the parameters in the proposed streaming downloader on the energy consumption.

1.4 Summary and Contributions

This work has mainly dealt with two broad problems: *i*) testing the energy performance of smartphone applications, and *ii*) evaluating the energy cost and designing an energy efficient downloader for smartphone streaming applications. The main contributions of this work can be summarized as follows:

1. generalizing the energy performance testing framework in [69] and giving a detailed flowchart that application developers can easily follow,
2. evaluating the energy cost of some popular streaming applications showing how download strategy that an application developer adopts may cause inefficient energy usage,
3. developing a model of an energy-friendly downloader for streaming applications and studying the effect of its parameters on the energy consumption,
4. giving a mathematical model for the proposed downloader and validating it by means of experiments.

The remainder of this work is organized as follows. In chapter 2, a comprehensive literature review for energy performance testing and delivering strategies for streaming applications is given. Chapter 3 introduces the generalized strategy for evaluating the energy performance of smartphone applications. The proposed model of an energy-friendly downloader for streaming applications and its mathematical representation are given in chapter 4, and chapter 5 concludes this work.

Chapter 2

Literature Review

The rapid growth of smartphones and internetworking has brought lots of network-related applications to App Stores. Many of these applications make use of Internet access, requiring considerably large amounts of energy. Since multimedia applications are known to be energy-hungry and mobile devices are energy-poor, having power-efficient multimedia streaming applications for handheld devices is important for prolonging the battery recharging time and improving user satisfaction. In order to evaluate and enhance the energy consumption of a smartphone application and detect any energy bugs, energy performance testing is required. This chapter reviews the literature in terms of *i*) energy testing for smartphone applications, and *ii*) enhancing energy consumption of streaming applications.

2.1 Testing Energy Consumption of Smartphone Applications

In the world of application testing, the number of possible inputs (parameters such as WiFi, 3G, GPS, Camera ...etc) for any application is very large if all of them are considered for a test. Exhaustive testing that considers all possible combinations of the test cases is desirable [27]; however, due to time and resource limitations, only a subset of all test cases can practically be considered as a test space [61]. Many techniques have been developed to reduce the number of test cases and yet improve the reliability and the performance of the application under test. Pairwise testing is a well-known technique that covers all

possible pairs of parameter states. For each and every pair of states that belong to different parameters, there is at least one test case that contains both of those states [35].

However, an application could misbehave only with three or more combinations. Study results found that all the known application misbehaviors are caused by interactions among six or fewer parameters [54]. Another technique called t-wise testing requires that every combination of any t parameter values must be tested at least once [29].

A more practical and efficient approach in examining applications than t-wise testing is variable strength interaction testing [30]. This technique focuses on a specific set of parameters and applies a higher strength testing to them without ignoring the rest. This is required because interactions do not often exist uniformly among parameters. Some parameters have strong interactions with each other while others may have few or no interactions.

All of the aforementioned techniques were proposed for functional testing. Their main aim is to maximize the chance of finding errors and bugs in the application under test. There is not much work that has been done on evaluating the energy consumption of smartphones applications. The work in [69] is one of the basic and important studies that were done in the area of energy testing of smartphone applications. In that work, a test selection technique was proposed to have a reliable parameter configurations among smartphones and to obtain the behavior of energy consumption for an application. The parameter that mostly affects the power consumption of the application under test is chosen as a primary parameter. The rest of the parameters are called standalone parameters. The combinations of the standalone parameters one-by-one with the primary parameter are considered as test space. Our work improves this methodology by considering more than one primary parameter.

2.2 Techniques to Save Energy in Smartphone Streaming Applications

Energy consumption in battery-based handheld devices that are wirelessly capable of multimedia streaming has been a significant research issue for the last decade. That is because streaming multimedia contents over wireless networks consumes lots of power for reception, decoding and displaying of the content. It has been shown that the energy cost of receiving a multimedia content over a WiFi interface is approximately three times more than the energy cost of decoding it [43]. Since our work mainly focuses on multimedia reception, this chapter reviews the publications regarding the energy cost of receiving multimedia contents

and omits the ones dealing with the energy cost of decoding and displaying multimedia contents.

Energy consumption of delivering multimedia content is affected by all layers of the protocol stack, ranging from the physical layer to the application layer. Lots of layer-wise and cross-layer approaches for energy saving have been proposed [66][43][50][36][62]. We briefly summarize the energy saving mechanisms according to the protocol layers.

2.2.1 Layer-Wise Mechanisms

1 - Physical Layer:

The PHY layer deals with data transmission/reception over wireless channels. It consists of radio frequency, modulation, power control and channel coding units. Using multimedia streaming applications, the radio interface is usually capable of a much higher data rate than the stream rate (including the overhead), and it is obvious that using the full capacity of the radio while only a small fraction of it is needed is impractical. Many techniques have been proposed to enhance the power consumption of the PHY layer.

Schurgers *et al.* [76][75] proposed a Dynamic Modulation Scaling technique (DMS) that dynamically adapts the modulation level to match the instantaneous traffic load and hence controls the power consumption. When the instantaneous traffic load is lower than the peak value of the radio component, transmissions can be slowed down, possibly all the way to the optimal operating point. This technique (DMS) is the exact counterpart of Dynamic Voltage Scaling (DVS), which has been shown to be extremely effective for processor power management.

Cui *et al.* [32] have shown that applying DMS that is based only on the streaming rate does not guarantee the minimal energy savings because energy consumption does not depend only on the transmission rate but also on the transmission distance. They show that for short transmission distance the lower transmission rate may not result in better energy savings due to the circuit energy consumption.

Multiple-Input Multiple-Output (MIMO) is a promising technique that could enhance energy consumption at the PHY layer [38]. However, MIMO is not always more energy efficient than Single Input and Single Outout (SISO) because MIMO requres more antenna devices that consequently consume more circuit power [59]. For low transmission rates, SISO is more power efficient than MIMO. However, for high transmission rates, MIMO outperforms SISO from an energy-saving point of view. A dynamically switching mecha-

nism between MIMO and SIMO has been proposed in [51]. It is shown that this switching mechanism is more energy efficient than MIMO.

2 - Link Layer:

Lots of solutions enhancing energy consumption at the link layer have been proposed [80]. Some of these solutions require modifications to the standard protocols. Since our interest is on WiFi interface, we summarize the important WiFi link layer solutions.

The IEEE 802.11 standard supports Power Saving Mechanism (PSM) to conserve energy [68]. The wireless station (STA) and the Access Point (AP) agree upon a beacon period. The AP buffers packets destined for the STA. The STA wakes up periodically to listen for the beacon. The beacon frame contains a Traffic Indication Map (TIM) that tells the STA if packets are currently being buffered at the AP. If the AP has buffered packets for the STA, the STA sends a PS-POLL frame to the AP, and the AP sends a buffered packet back to the STA. If the *MORE* flag in the received packet is set, the STA continues to PS-POLL till no more packets are buffered at the AP. The PSM approach saves energy by keeping the radio interface off except during the beacon period where the STA briefly wakes up to communicate with the AP. However, the PSM adds a delay in between beacon periods when the radio is off. This delay affects real-time applications such as live-streaming and VoIP. Furthermore, buffering packets at the AP may cause the TCP to estimate higher Round Trip Time (RTT) and consequently the TCP congestion-control mechanism will significantly slow down the traffic rate in TCP-based streaming applications [43][53].

Because of the latency inherited with the static PSM approach, an Adaptive PSM (PSM-A) that dynamically switches between Continuous Awake mode (CAM) and PSM has been widely adopted [53]. In PSM-A, the device switches between PSM and CAM based on some heuristics (e.g. reception of a threshold number of packets or lack of network activity for a pre-defined duration). The device informs the AP of its transitioning to PSM or CAM by sending NULL data frames with the power management bit set to 1 or 0, respectively. It has been shown that the overhead cost of switching the radio interface between CAM and PSM is trivial [79]. However, since the device remains in CAM for an idle timeout period after every CAM to PSM transition, for applications with intermittent network activity, PSM-A is more energy inefficient than static PSM [33].

In order to enhance power savings while supporting QoS-sensitive applications such as VoIP, multiple power saving protocols have been proposed in the family of 802.11 standards. In 802.11e [1], an Automatic Power Save Delivery (APSD) protocol is proposed. It

introduces two modes based on the delivery mechanism, namely, the unscheduled APSD (U-APSD) and the scheduled APSD (S-APSD). In U-APSD, if a mobile station is informed that it has data buffered for it, it issues a trigger frame to the AP and the AP responds with the frames buffered for it. In S-APSD, the unicast pending data are scheduled periodically by the AP to deliver at a designated time, given that the mobile station has agreed with the AP about the scheduling intervals. Both modes aim at improving the energy utilization using bursty frame transmissions. However, the uncoordinated trigger frames can cause the mobile station to wait for an excessively long time before the data transmission commences [71]. Meanwhile, the S-APSD depends on the resource reservation per stream by negotiation between AP and the mobile station, which degrades the flexibility compared to U-APSD and is not fit for the traffic with a variable bit rate. Further, none of them tackle the issue of multicast data.

In the 802.11n [2], the APSD is enhanced by a Power Save Multi-Poll (PSMP)procedure. Similar to APSD, the PSMP procedure can be triggered by a frame (U-PSMP) or scheduled periodically by the AP (S-PSMP). It starts with broadcasting a PSMP message by the AP to the network which contains the scheduling information for downlink and uplink transmissions. The mobile station is assumed to wake up to receive this message and arranges its network activities and power state transitions based on the scheduling information. The PSMP creates effective transmission periods for broadcast, multicast, and unicast data, and the power consumption is greatly reduced by periodical occurrence of the PSMP procedure and its scheduled frame transmissions. However, compared to the static PSM, the overhead of the management frames is non-trivial and the mechanism to maintain the uplink transmissions is fairly complex.

The 802.11v [3] introduces a Flexible Broadcast Multicast Service (FBMS) to provide flexible management over the multicast traffic in power saving protocols. In FBMS, a multicast stream can choose to be served at a configurable delivery interval, Thus, traffic of different multicast groups can be differentiated by selecting different delivery intervals. The mobile station does not have to pay extra energy to receive irrelevant multicast traffic. The major disadvantage of the FBMS is that it significantly increases the delay of the multicast streams, and the energy saving is reduced when the number of multicast streams increases; as in this case, some streams have to be delivered within the same periods.

Besides these standards, lots of solutions requiring modifications to the standard link layer protocols are proposed. Many of these solutions are based on prediction policies. Some history-based prediction strategies have been proposed in literature such as the work in [25]. The authors show that streaming traffic tends to be regular and predictable. They calculate the next sleep interval as the average n earlier idle times. Such strategies result in good energy-saving. However, for some streaming formats, there was relatively high

packet loss. This led the authors of [81] to develop a linear prediction-based time series technique. They showed that a linear prediction-based approach is more energy-efficient than a history-based approach. However, because many media streaming techniques exhibit high variation in the data packet sizes and inter-packet arrival times, it is hard for the prediction algorithms to reliably predict the lengths of no-data intervals.

Liu *et al.* [58] inspired by the mismatch between the high bandwidth of the 802.11 interfaces and the modest data rate requirements of many popular network applications, and presented micro power management (μPM) to reduce the idle time (the timeout of PSM-A) even when the idle time is less than $200ms$. μPM does this by relying on a link layer retransmission mechanism for frames missed during a short nap and by predicting frame intervals based on history in order to limit the frame delay. Experimental results with μPM implementation demonstrate up to 30% power savings for multimedia streaming with a very low frame loss rate.

He and Yuan [42] pointed out that the power consumption of a PSM client depends not only on the traffic load destined for it, but also on the traffic loads destined for other clients. The reason is that when the access point serves more than one client, the packet transmission from the access point to the PSM client may be delayed by data transmission to/from other hosts. This results in the prolonging of the waiting time for the PSM client to receive the buffered data, and thus more power consumption. For this reason, they further scheduled packet transmission for PSM clients using precise timing slots. Their TDMA-like scheduling algorithm eliminates contentions and achieves near optimal power saving for the mobile hosts.

3 - Network Layer:

The main functions of the network layer are routing packets and congestion control. In wireless mobile networks, the network layer has the added functionality of routing under mobility constraints and mobility management including user location update, etc.

In [70], the author proposed an information-centric approach to networking, which allows multimedia streaming to be transformed from an energy-heavy network service to a lightweight one. Unlike the current host-centric mobile networking paradigm based on end-to-end always on connectivity, an information-centric approach could be used to reduce the time network interfaces have to remain active. Such mobile networking architecture will bring in new toggles that allow users, operators, and service providers to make their own communication/computation/storage trade-offs and enhance the energy consumption.

However, the information-centric approach would require an adoption of a new networking paradigm and a change in the way the information is distributed to its intended recipients.

4 - Transport Layer:

The transport layer provides end-to-end delivery services for applications. There are two most common transport layer protocols that streaming applications might use: Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). Although traditionally UDP is the ideal protocol for streaming delivery, today TCP-based streaming accounts for more than 90% of the Internet streaming traffic due to *i*) the easiness of deploying TCP/HTTP streaming applications , and *ii*) the wide use of NAT and firewalls that are not UDP-friendly [84][41][18][79]. Because this work concerns TCP-based streaming, we focus on the main TCP-based solutions that appear in the literature and omit UDP-based solutions.

Effectively saving energy during a TCP streaming session is difficult because TCP traffic tends to be smooth, leaving little potential for the WNIC to switch to sleep mode. The basic idea behind most TCP techniques is that the client increases the amount of time that can be spent in sleep mode by shaping the traffic, i.e forming bursts out of constant or variable bit rate stream and scheduling them in such a way that the existing link layer PSM can be beneficially used.

Bertozzi *et al.* [22] proposed a way that modifies the TCP flow control mechanism in order to generate TCP idle periods. They utilize two orthogonal cases in which TCP becomes inactive: when the TCP buffer is full and when the TCP buffer is empty. In the case where the buffer is full, the client advertises zero window, and the window is never open till the buffer is completely drained by the application. The WNIC can go to sleep during this period (between the full buffer and empty buffer conditions). In the case where the buffer is empty (e.g. no active sockets or temporarily un-utilized sockets), if there is no activity for a timeout, the WNIC is switched off, and it is switched back on when a new TCP packet transmission or a read operation from the TCP buffer has to be carried out. Their experimental results indicate energy savings between 28% and 69% compared with static PSM. However, this approach increases the number of re-transmission attempts as well as transmission delays.

A client-centric scheme [82] is proposed to reshape the TCP traffic into bursts, and put the WNIC to sleep between two bursts by modifying the client TCP stack. The basic idea is that the client forces the server to send each window of packets in a burst so that potential sleep time between bursts is maximized. The client monitors the TCP receiving

buffer. Zero window TCP-ACK or full window TCP-ACK is sent from the client to the TCP server depending on the condition of the receiving buffer of the client. Although this scheme has the potential to save energy, it increases the data transmission time as a trade-off, which can be a high cost for some bulk data transmissions and unacceptable for streaming media applications with stringent QoS demands.

It has been shown that the power consumption of streaming data at a mobile host is negatively affected by the congestion (and consequently the TCP congestion control) occurring at the wired network. The wireless single-hop link between the mobile host and the access point usually achieves a much higher bandwidth. Thus, the WNIC spends most of the time in the idle state waiting for the congested packets to arrive. For that matter, the work of [15] and [12] proposes an alternative to the legacy TCP protocol for the management of communications between mobile hosts and fixed hosts: the indirect-TCP. It divides the end-to-end TCP connection between the mobile host and the fixed host into two connections, one between the mobile host and the access point, the other between the access point and the fixed host. At the access point, a Power-Saving Transport Protocol (PS-TP) daemon is in charge of transferring the traffic between the two connections. The link between the mobile host and the access point is a single-hop link; therefore, some functionalities of the classical TCP such as the congestion control mechanism are not required in this connection. A Simplified TCP (STP) is used between the the mobile host and the access point, and the legacy TCP is used between the access point and the fixed host. This approach saves energy by reducing the total transfer delay and the amount of time the WNIC is active.

5 - Application Layer:

Energy efficiency at the application layer is becoming an important area of research as is indicated by industry. Many APIs and power management analysis tools are being developed to assist software developers in creating programs that are more power conserving [66]. Lots of research is being done in application layer energy saving [66]. However, since this work is solely interested in content-delivery techniques for energy conservation, other techniques such as computations offloading, adaptive encoding rate, encoding schemes, and GUI design are ignored here.

A good amount of research has proposed proxy-based solutions at the application level. A proxy can enable a mobile host to save energy by making downloaded stream bursty so that the intervals between bursts are long enough to let the communication interfaces sleep. Hoque *et al.* [44] tackled the energy consumption of a TCP-based multimedia streaming application using a traffic shaping proxy. Their approach relied on the default

PSM available in the mobile host. The proxy is placed between the mobile client and the server. Once the proxy receives a request from the client, it establishes a connection to the server and forwards the client's request to the server. The proxy repeatedly buffers the constant bit rate traffic received from the server for a fixed period of time and forwards the buffered data to the client in the form of bursts. They showed that using a proxy can save up to 65% of the energy consumed if using only the default PSM. They also found that the location of the proxy has an impact on the energy savings. Their results show that using a proxy which is close to the client can gain around 40% more energy savings than using one further away.

The authors of [37] presented a scheme, called *CatNap*, that saves energy by combining tiny gaps between packets into meaningful sleep intervals using a proxy at the access point. *CatNap* (*i*) utilizes an application-level proxy to decouple the wired segment from the wireless segment, (*ii*) estimates the available bandwidth in the wired and wireless segments (the wireless segment usually has a much higher bandwidth than the wired segment), and (*iii*) schedules the segments to be transferred to the client in a burst with high speed but in a short time without increasing the overall transfer time. Streaming applications provide the proxy with workload hints specifying the data block size and the willingness to tolerate delay. *CatNap* proxy uses these hints to schedule delivery of the segments. According to the experimental results, *CatNap* allows the WNIC to sleep for up to 70% of the total transfer time.

Korhonen *et al.* [52] studied the impact of the burst length and peak transmission rate for observed packet loss and delay characteristics. They then proposed an adaptive streaming technique for the UDP-based multimedia streaming. The system works at the server or a proxy and manipulates the burst interval depending on the packet loss and network situation experienced by the streaming client. Their experimental results show that this adaptive streaming technique can achieve up to 46% energy savings

Zhang and Chanson [85] suggested two proxy-based scheduling algorithms for multiple streaming mobile hosts. In the first algorithm, the proxy is oblivious to the host's power characteristics and schedules a host with the lowest burst reception time with the highest priority. The second algorithm is based on a heuristic approach that considers the host's idle power consumption and battery state. The first scheme minimizes the sum of times required by all the clients to finish streaming media contents, while the second scheme exploits battery information of all the clients to minimize the weighted sum of the times needed. The drawback of the first algorithm is that the client with the highest reception time experiences the longest waiting time and therefore consumes more energy, whereas the second algorithm lets the client with the highest battery capacity experience the longest waiting time and consequently consume more energy.

Chandra and Vahdat [26] proposed an application-specific proxy-based system that consists of a client side proxy and a server side proxy. The server side proxy informs the client side proxy of the next arrival. The client then can sleep between data transfers. Although it seems promising, this scheme is not compatible with the IEEE 802.11 standard. Mohapatra *et al* [65] proposed a proxy-based traffic regulation mechanism to enhance energy consumption by the network interface. Their scheme dynamically adapts to changing network and device conditions. The proxy buffers the data and transmits it to the host as optimized bursts along with control information. Rosu *et al.* [74] focused on using an HTTP proxy to shape incoming traffic to create longer idle periods, thus enabling wireless devices behind the proxy to sleep for longer periods of time.

Since a server-client relationship can be seen as a producer-consumer relationship, a buffer can be utilized to enhance the energy consumption by creating idle periods wherein the producer or the consumer can go to sleep (without buffers, the consumer and the producer are always busy) [24][23][72]. In [24] the authors presented a general method to enhance power consumption using a data buffer and calculated the optimal buffer sizes for constant producing rates and consuming rates. This work is extended in [73] by considering variable consuming rates and including the delay needed to awaken the producer. The results show that using this approach can save up to 74% of the energy for video streaming applications. Cia and Lu [23] further enhanced this work by providing a general approach for energy management based on buffer insertion and the concept of inventory control which is a scheduling problem . In this work, the authors proposed a method to find the minimum-energy schedules and the corresponding minimum buffer size. In [72], the authors farther investigated energy conservation using buffer insertion. They proposed analytic formulas for obtaining the optimum buffer sizes for application-specific QoS without heuristic decisions. Their simulation results show that the average buffer size for LAN and WAN adapters is about 16MB.

Another approach to shape streaming traffic is by utilizing a buffer at the server side. Adams and Muntean [8][9] proposed a buffering technique that exploits the inherent power saving mode of IEEE 802.11. Data destined to a certain mobile client is hidden and thus not made available to it for multiple beacon intervals. The authors estimated that using this scheme increases the battery life 160% compared to static PSM. Also their tests show that packet sizes and intervals between packets have a significant effect on the power consumption.

Li *et al.* [57] investigated the effect of the TCP keep-alive and zero-window probes in HTTP video streaming and user viewing behavior on power consumption. They found that the TCP zero window probes and the corresponding keep-alive acknowledgments bring the WNIC into the active mode. Even though these TCP segments are very small and last for a

short time, they keep the WNIC awake for a longer time waiting for the timeout to elapse. They studied this in 3G and 4G network interfaces where the timeout (the Tail state) is relatively long. They also found that most video sessions are short, and users tend not to watch the whole video content. Based on their statistics, 80% of YouTube sessions are less than half of the corresponding video durations. Motivated with these findings, they proposed *GreenTube* [57], a system that reduces the power consumption of smartphone YouTube streaming over 3G and LTE networks. *GreenTube* achieves that by closing the TCP connections in between bursts to prevent zero-window probes and by adapting burst size to user viewing behavior (based on history) and network conditions. Experimental results show that *GreenTube* achieves power savings up to 70% in 3G and 4G networks.

2.2.2 Cross-Layer Mechanisms

Standalone traffic manipulation techniques at layers might not be adequate to provide significant energy savings. Thus, lots of cross-layer techniques have been proposed to improve the energy savings of delivering multimedia streams to mobile hosts. Since this work is mainly focused on the application layer, in this section we review cross-layer techniques wherein the application layer is involved.

Bertozzi *et al.* [21] proposed a technique that is based on an application level buffer. During multimedia streaming, the link interface is shut down whenever the application buffer is full, and it is turned on when the buffer is reduced to a threshold level. During the disconnection period, the access point buffers the incoming stream for the mobile device and thus creates bursty traffic. However, turning on the Wi-Fi interface is a high energy consuming process, because it requires powering up the chip, re-associating with the AP and in turn re-establishing the connection (for TCP-based streaming). Their simulation results show that the minimum buffer size is a function of the network bandwidth, and they suggested that buffers should be as large as possible. By using large buffers, the awakening overhead energy is reduced across a longer interval. Bagchi [20] implemented another multimedia streaming technique based on a fuzzy logic adaptive buffering algorithm. In this technique, a client player would pull streaming traffic from the streaming server via the WiFi interface according to the client player buffer status, variations of network bandwidth, and required QoS. This approach also switches off the WiFi interface in-between bursts to save energy. It is illustrated through experimentation that this technique reduces data pause delay and data loss while enhancing energy savings.

In [6], Acquaviva *et al.* proposed two Dynamic Power Management (DPM) techniques to control the playback buffer and thus power consumption. In the first technique called

Closed-Loop DPM, a client informs its server when to send and not to send data based on the buffer status. The server continues refilling the client's playback buffer. When the playback buffer reaches the high-water mark, the client tells the server to stop sending and shuts down the WNIC. When the playback buffer reaches the low-water mark, the client wakes up the WNIC and informs the server that the low-water mark has been reached so that the server can start refilling the client's playback buffer. In the second technique called Open-Loop DPM, the server predicts when the client buffer will be empty by exploiting its knowledge of the workload. The server decides the burst size and the time between bursts to fully exploit the playback buffer at the client side. Each burst of data transmitted by the server is preceded by a special control packet carrying information about the number of packets in the burst and the transmission time of the next burst.

Anand *et al.* [11][10] implemented a Self-Tuning Power Management (STPM) module which dynamically adjusts the power management mechanism for wireless interfaces using application hints. STPM operates at the Operating System level, and exploits hints provided by the applications. Essentially, hints describe the near future requirements of applications in terms of networking activities, such as next-transmission time, maximum delay-tolerance, number of transmissions and type of transmission (foreground or background). STPM exploits these hints, and the energy characteristics of the entire system, to manage the wireless interface appropriately. When these hints are not available, STPM estimates the traffic features by spoofing it. Anastasi *et al.* [13] further proposed a generic architecture, called Cross-Layer Energy Manager (XEM). XEM also exploits application behavior. However, unlike the STPM scheme that switches the WiFi interface between PSM mode and CAM mode, XEM switches the WiFi interface between PSM mode and off mode according to the traffic observation. Their results show that the XEM scheme can save from 20% to 96% compared to the standard PSM. Howevr, it is not suitable for interactive applications and multimedia streaming because of the inherent delay.

Acquaviva *et al.* [5][7] proposed a server controlled power management scheme where the server exploits the workload, traffic-related information and feedback from the mobile client to minimize the energy consumption of the mobile network interface card. Their approach involves a Client Power Manager (C-PM) running on the client device and a Server Power Manager (S-PM), running on the server. The two managers exchange power control information through a dedicated TCP connection. C-PM obtains the buffer size, depletion rate and transition time between on and off states for client's WNIC and sends this information to the S-PM. Based on the information the S-PM gets from the C-PM and the traffic conditions, the S-PM decides when to enable the client's WNIC power saving mode and schedules transmissions to the client in bursts (The burst size and delay are computed at the S-PM). Their experimental results show that their strategy saved 67% of

the energy as compared to CAM, and 50% as compared to the standard PSM.

Radu *et al.* [31] proposed an annotation-based technique that is different from other historical or statistical prediction based techniques. Annotations specifying data size and time of packets are generated at the server side (on-the-fly or offline). These annotations are embedded into the media stream (or transmitted over a separate channel) and sent to the client. On the way, annotations are used by the access point to shape traffic into bursts before forwarding it to the client. Using the information in the annotations, the access point can accurately estimate transfer rates and buffer network packets with a minimal overhead. Since packets are transmitted in bursts to the client, the WNIC is switched into sleep mode between the burst transmissions. The card is switched back to active mode just before the next burst arrives at the client side. A buffer is used at the client side to allow smooth playback. Simulation results show that this annotation-side technique can save up to 75% more energy than the standard PSM. The authors also showed that this approach is not susceptible to packet drops occurring in prediction based techniques due to prediction errors.

Mohapatra *et al.* [63][64] proposed a proxy-based technique that utilizes a dedicated control channel used to exchange information/commands between the proxy and the mobile host. The proxy utilizes the feedback knowledge sent by the client, the bandwidth availability and the attributes of the wireless access points and accordingly transmits optimized bursts to the client. Even though packets are transmitted in bursts by the proxy, the device receives packets that are skewed over time due to the the fair queueing algorithms implemented at the access point. Their mechanism optimizes the stream, such that the optimal video burst sizes are sent for a given noise level. The excremental results show that this mechanism can provide 50% energy saving. However, the proposed mechanism will probably not perform well in light network application due to the overhead network usage by the control channel especially in mobile phones where the network interface is the major energy consumer [67].

In [16] the proxy idea along with the playback buffer is again implemented, introducing the novel Real Time and Power Saving (RT-PS) protocol for the communication between the server and the mobile host. The proxy is responsible for planning bursty transmissions to the client and warns the client of a non-transmission period when it starts. It also notifies the client of the duration of the next non-transmission period so as to switch the network interface to sleep mode for that entire period. The authors proposed that plans of transmission are dynamically obtained by considering an estimate of the available bandwidth on the wireless link along with the current client buffer level. A non-transmission period ends when the client buffer level falls below a threshold, a low-water mark, that is dynamically changed according to the available bandwidth. Along with the media stream, the

proxy and the client host exchange command and information packets. Their experimental results show that implementing this RT-PS protocol can save up to 90.83% energy. The RT-PS protocol is further studied in [14]. The authors implemented the RT-PS protocol running on top of UDP between the mobile host and the access point for audio streaming. The client-side component of the RT-PS protocol *i*) sends the request for an audio file to the proxy at the access point specifying its playback buffer, *ii*) estimates the throughput of the wireless channel and notifies the proxy, *iii*) switches the WNIC to sleep when a sleep command is received from the proxy and switches WNIC back to active mode when the sleep time is elapsed. The proxy-side component of the RT-PS protocol *i*) transmits the audio frames to the client during the active periods, *ii*) keeps track of the WLAN bandwidth and the level of the user buffer, *iii*) decides whether or not to stop transmitting and calculates the sleep time. The authors performed experiments to show that this technique allows energy saving from 76-91% of the total consumption. They also show that the fraction of frames discarded by the proxy (those frames were estimated to arrive late for playback) or lost during transmission was below 5% of the total frames which is acceptable for audio playback.

Shenoy and Radkov [77] proposed two proxy-assisted techniques. In the first technique, a proxy sends multimedia traffic at a periodic interval to a client and the client uses a history-based traffic prediction mechanism to power on and off its WNIC interface. In the second technique, the proxy forwards the traffic stream at a periodic interval and sends one control packet to indicate the next arrival time of the future packets. A mobile host uses this information to switch on the interface on time. Their experimental results show that their proxy-based technique where control packets are used can reduce the potential energy wastage at the network interface by 65-98%.

Gundlach *et al.* [40] developed a transparent proxy to schedule both TCP web traffic and UDP-based bursty streaming traffic to all active clients in a WiFi network. The proxy broadcasts the new schedule prior to a burst and after all the clients have received the data of the previous schedule. The proxy marks the type-of-service bit in the IP header of the last packet so that the client knows when to transition its WNIC back to low-power mode. The proposed transparent proxy collects client addresses using ARP spoofing in order to achieve transparency, so it must be implemented within the WiFi network. The experimental results show that about 53-77% power saving with less than 2% packet loss can be achieved using this scheduling-based transparent proxy.

A very recent work [45] has studied the relationship between burst size and power consumption for TCP-based multimedia streaming when traffic shaping is used. The authors modeled the power consumption of bursty streaming traffic over TCP by considering two scenarios: *i*) the total buffer size is bigger than the burst size; and *ii*) the buffer size is

smaller than the burst size. They found that burst size that equals the sum of the application playback buffer and the TCP receive buffer gives the minimum power consumption. The authors also proposed a proxy-based energy-aware multimedia streaming system called EStreamer. In this cross-layer approach, a proxy keeps track of the received TCP acknowledgments coming from the client. Once the proxy encounters zero window advertisement, it tunes the burst size accordingly in order to get a burst size that just fits the sum of the TCP buffer and the playback buffer. Their experimental results show that using EStreamer can achieve energy savings up to 65% with WiFi interface, 50-60% with LTE interface, and 35% with 3G interface. A similar recent work [78] also used EStreamer to study the power consumption of multimedia streaming over 3G and LTE networks. The authors showed the effect of different network configurations on the achievable energy savings. They also studied the impact of background traffic (application level signaling between server and client) and found that the background traffic for YouTube applications can increase the average current up to 27%.

Yan *et al.* [83] proposed a client-centric technique to generate bursty traffic for TCP based applications in order to save energy. The basic idea is to modify the client's TCP receive window size to zero periodically at the network layer. Once the window size is set to zero, the server-side TCP cannot send packets to the client, and the client can turn its WNIC interface off for this idle time. When the clients receive window size is restored; the TCP at the server sends all the data in the congestion window resulting in a traffic burst. This technique leverages the TCP timestamp option at the client-side in order to estimate the Round-Trip-Time (RTT). In order for the client to detect the end of a burst, the client sends a window probe packet to the server immediately before it sends a close window acknowledgment, and the server will respond with a probe response packet which marks the end of a burst. Their experimental results show that this client-centered technique can achieve about 21% energy savings compared to standard PSM.

The idea of shaping traffic in order to reduce power consumption is not novel. Lots of traffic shaping solutions have been proposed at the link layer to create opportunities for WNIC to shut down. However, application-level traffic shaping approaches can create long idle periods for the WNIC with less-frequent on-off mode switching, so that the high shutdown transition cost (in terms of latency and energy) can be fully amortized and power can be saved during long idle times. Lots of the proposed solutions at the application level are based on the following approaches: i) shaping the traffic at the server ii) shaping the traffic at the proxy iii) shaping the traffic at the client. In addition to the easiness of implementation, client-centric traffic shaping has the advantage of knowing how the power hungry components are being used and consequently can save energy without overhead traffic. Most of the client-centered approaches found in literature are either history-based,

prediction-based or buffer-based. However, besides prediction errors, history-based and prediction-based approaches require assistance from the server or a proxy. A client-centric buffer-based approach does not require any assistance from network infrastructure and provides good energy savings with smooth multimedia playback. Unlike our approach, to the best of our knowledge, all the buffer-based approaches found in literature either are cross-layer approaches or discard the standard PSM altogether and propose customized scheduling solutions for the MAC layer. Our client-centric buffer-based approach does not require any cross-layer communication, and it relies on the standard Adaptive Power Saving Mechanism (PSM-A) implemented in most of today's hand-held devices.

Chapter 3

Testing Strategy

In this chapter, methodologies and tools used to evaluate the energy consumption of smartphone network-related applications are presented. More specifically, a framework to assess the energy consumption and find energy bugs of applications is described in section 3.1, and section 3.2 presents the method we use to capture the network traffic wirelessly.

3.1 Energy Assessment Framework

3.1.1 Introduction

The requirements in energy performance testing are different than those in functional testing. Whereas functional testing most likely treats the parameters (the input) equally and fairly, energy performance testing puts more weight on particular parameters that mostly impact the power consumption. In functional testing, all parameters are equally expected to cause a bug; however, in energy performance testing, the parameters that control the hardware (e.g. GPS) are more expected to cause energy bugs. Also, in energy performance testing, the relationship between parameters (dependency, for example) can be considered to reduce the number of required test configurations.

Due to the enormous number of phone configurations and application contents, an immense number of test cases exists [19]. Therefore, a selection technique reducing the number of test cases is desired. For this reason, a framework to select user-level test cases for energy cost evaluation of smartphone applications [69] has been proposed. In the

current work, we have generalized the methodology in [69] and explained it by means of a detailed flow chart.

The concepts of parameters, configurations, application contents, and test cases are at the core of this methodology. An explanation of these terms is as follows:

- *Parameters:* The settings of smartphone parameters affect the power consumption of applications. For example, the power consumption of a music player application is affected by the level of the device volume. Most of the parameters have various states representing different values. Some of those parameters are display size, Bluetooth, and brightness level. Table 3.1 shows some parameters with their states for five different smartphones, namely, *BlackBerry 9700*, *HTC HD2*, *HTC Nexus One*, *Nokia E71*, and *iPhone 3GS*. In the table, 'Yes' means the parameter is available on that particular smartphone, 'No' implies that the parameter is not available, and 'Alternative' denotes that a similar parameter is available. Some parameters take discrete values, and others take continuous values.

- *Configurations:* A configuration of a smartphone is an instance of all parameters for which a device is used to run an application [19][69]. Since the total number of all possible combinations of parameters' values is vastly large, there is an enormous number of configurations.

- *Application contents:* Application contents represent the settings of the application under test. Since this is an energy performance testing, only application contents that affect the energy performance metric are considered. For example, a video playing application supports different file formats (contents), and these different file formats require different amounts of processing time. Also, they use different amounts of memory. As a result, the type of video file affects the power consumption of a video player application.

- *Test cases:* A test case represents input and output of an experiment. A device configuration and application contents are the input of the test case. The output is the power consumption. We are not interested in the user output (the service given by the application). We observe the power consumption during the execution time and consider it as a test output. Since there is an enormous number of configurations and a large number of application contents, the number of possible test cases is very large.

Group	B_i	Parameters	Description	BB9700	HTC Nexus One	Nokia E71	HTC HD2	IPhone 3GS
Basic Group G_0	01	Display	Size of display	480 x 360 pixels, 2.44”	480 x 800 pixels, 3.7”	320 x 240 pixels, 2.36”	480 x 800 pixels, 4.3”	320 x 480 pixels, 3.5”
	02	Operating System (OS)	Name of the OS	BlackBerry OS	Android	Symbian	Windows CE	iPhone OS 3
	03	Battery Capacity	Battery type and capacity	Li-Ion 1500 mAh	Li-Ion 1400 mAh	Li-Po 1500 mAh	Li-Ion 1230 mAh	Li-Ion 1250 mAh
Active Group G_1	31	Volume	This option allows the user to change the volume level of the device	Yes (0,1,_,10)	Yes Option 1: Sounds (Silent) Option 2: Volume levels: (0,1,_,15)	Yes volume levels (0,1,_,10)	Yes Option 1: Sounds (Silent) Option 2: Volume levels (0,1,_,15)	Yes volume levels (0,1,_,16)
	32	Brightness	This option allows the user to change the brightness level of the device	Yes (0, 10,_,100)	Yes Continuous (0 to 100%)	Yes (0, 25,_,100)	Yes Continuous (0 to 100%)	Yes Continuous (0 to 100%)
	33	Bluetooth	This option allows the user to turn on/off their Bluetooth connection	Yes (ON/OFF)	Yes (ON/OFF)	Yes (ON/OFF)	Yes (ON/OFF)	Yes (ON/OFF)
	34	Data Access Mode	This option allows the user to select from WiFi/EDGE/3G connections	Yes	Yes	Yes	Yes	Yes
Passive Group G_2	61	Network Selection Mode	This option lets the mobile device to select the network manually or automatically	Yes (Auto/Manual)	Yes (Search automatically)	Yes (Manual/Auto)	Yes (Auto - Select/Deselect)	Alternative (Auto – Select / Deselect)
	62	WiFi Settings - Network Notification	This option prompts the user whenever any WiFi network is available	Alternative Prompt when manual connection or login is required	Yes (ON/OFF)	Yes Option 1: Show WiFi availability (Yes/No) Option 2: Scan for Networks (every 1 to 10 min)	No	Yes (ON/OFF)
	63	Portable WiFi Hotspot	This option leads the mobile to act as a WiFi hotspot	No	Yes Portable WiFi Hotspot (Select/Deselect)	No	Alternative Internet sharing (Select/Deselect)	Alternative (Setup Internet Tethering)
	64	Screen Timeout	This option allows the user to set the display timeout for the screen	Yes (10 sec to 2 Min)	Yes (15 sec to 30 min)	Yes (5 to 90 sec)	Yes (1 to 10 min on battery power) and (1 to 30 min on external power)	Yes (1 to 5 min or Never)
	65	Automatic Dim Backlight	This option dims the display light automatically	Yes (ON/OFF)	No	Yes (5 to 60 sec)	Yes (10 sec to 5 min on battery power) and (1 to 10 min on external power)	No
	66	Security - Firewall	The user can enable or disable the Firewall	Yes (Enabled / Disabled)	No	No	No	No

Table 3.1: Some of smartphones’ parameters.

3.1.2 Generalized Approach

The work in [69] is one of the basic studies that were done in the area of energy performance testing for smartphone applications. In that work, a methodology of test selection technique was proposed to have a reduced number of configurations which is used to obtain the energy behavior of an application. Since this work mainly generalizes the work in [69], a brief summary of [69] is given next.

Assume that there are m parameters in a smartphone d grouped in the set $G^d = \{B_1, B_2, \cdots, B_m\}$. A specific state of a parameter B_j is b_j^l, and it has $\eta(B_j)$ states. For example, the Volume parameter B_{31} for *HTC Nexus One* smartphone (refer to Table 3.1) has a state b_{31}^1 = '0', and the number of states for this parameter is $\eta(B_{31}) = 16$. A configuration β_i represents an instance of all settable parameters of a smartphone. A smartphone application A_i has a type of content C_i. The number of contents for a particular application A_1 is denoted by $\eta(C_i)$. For example, a music player application A_1 supports only two file formats: c_1^1 = 'MP3' and c_1^2 = 'WAV'. The number of contents for this application is $\eta(C_1) = 2$.

The number of test cases N_c is expressed as:

$$N_c = S_d \times \sum_{i=1}^{M} X_d(A_i) \times \eta(C_i), \tag{3.1}$$

where S_d is the number of configurations for a smartphone d. M is the number of chosen applications. $X_d(A_i)$ indicates whether or not the application A_i is available in the phone d.

The number of all possible configurations for a smartphone d is expressed as:

$$S_d = \prod_{j=1}^{m} \eta(B_j) \tag{3.2}$$

This number is extremely large for any smartphone [69], and it is not feasible to consider all the configurations of a smartphone while testing. To reduce the number of configurations, firstly the smarphone parameters are categorized as follows:

- *Basic Parameters* (G_0^d): This group contains all the parameters whose values are fixed. A user cannot change or adjust the values of those parameters. Processor,

memory and size of display fall into this group. Table 3.1 shows some basic parameters of the five smartphones mentioned earlier. Those parameters are kept aside during the testing process. However, when two or more smartphones are compared from an energy performance viewpoint, the differences in these parameters must be considered during the analysis of the test results.

- *Active Parameters* (G_1^d): This group contains the parameters of which the user can control and adjust their values. Basically, these parameters are some utility programs on top of the operating system (OS), which control hardware components. Applications for controlling the volume of a device and the brightness of the display are examples of active parameters. Table 3.1 illustrates some of the active parameters. The active parameters are further divided into two subgroups:
  - *Primary Parameters* (G_1^p): A subset of active parameters are included in a set named as primary parameters. This subset of active parameters appears to consume a significant proportion of the total power consumptions. So, they (the primary parameters) strongly impact the power consumption of an application, and their variations yield more informative test results. Those parameters are given more weight when selecting test configurations.
  - *Standalone Parameters* (G_1^s): These parameters are the rest of the active parameters. With the primary parameters, standalone parameters are considered for performing standalone configurations.
- *Passive Parameters* (G_2^d): This group contains the rest of the parameters whose values are also adjustable by the users. They are also utility programs incorporated in the OS or from third parties. For example, WiFi Hotspot and Bluetooth Discoverable are programs which use WiFi and Bluetooth interfaces, respectively. They do not control the hardware components, but they use hardware components, and thus, they affect the test outcomes. During the testing process, we need to take out the effect of these applications by keeping their values fixed throughout the testing process. Table 3.1 shows some of the passive parameters.

Only the active parameters are considered for the device configurations. The basic parameters are omitted and the passive parameters are fixed to certain values. The number of active parameters is m_A. The total number of test cases is given as

$$N_C = (S_d^P + S_d^S) \times \sum_{i=1}^{M} X_d(A_i) \times \eta(C_i), \tag{3.3}$$

where S_d^P is the number of primary configurations, and S_d^S is the number of standalone configurations. Primary configurations are obtained by considering primary parameters only. Standalone configurations are obtained by considering standalone parameters one by one with the primary parameters.

The work in [69] considers only one parameter in the primary set. However, for some applications, there may be more than one parameter highly impacting the energy consumption of a smartphone. For example, location related applications usually rely on the GPS parameter as well as the network access mode parameter. Therefore, it is important to consider multiple parameters in the primary set. This work considers the general case where there are $k \leq m$ parameters in the primary set.

The dependency between parameters is a key idea in our methodology. Suppose the parameters $B_i = \{b_i^1, b_i^2 \cdots, b_i^I\}$ and $B_j = \{b_j^1, b_j^2 \cdots, b_j^J\}$ are active parameters, and $\theta(b_i^u, b_j^q)$ is the energy cost of a test case where the parameter $b_i^u \in B_i$ and the parameter $b_j^q \in B_j$. The parameters B_i and B_j are independent of each other if their energy costs are additive; that is:

$$B_i \perp B_j \iff$$

$$\left[\left(\theta\left(b_i^u, b_j^q\right) - \theta\left(b_i^v, b_j^q\right)\right) \simeq \left(\theta\left(b_i^u, b_j^r\right) - \theta\left(b_i^v, b_j^r\right)\right)\right]_{u \neq v, q \neq r}$$

In order to check the dependency between two parameters, the energy costs of at least four test cases are required. Suppose the set G_1^p, a subset of G_1 ($G_1^p \subseteq G_1$), represents the set of primary parameters for smartphone d. In order to obtain general expressions for a number of configurations, we make the following assumptions:

- The set G_1^p is called to be independent if all parameters in this set are independent of each other. The notation $Ind(G_1^p)$ means that the set G_1^p is an independent set, while the notation $Dep(G_1^p)$ means G_1^p is a dependent set.
- A parameter B_j is independent of the set G_1^p if it is independent of all the parameters in that set. The notation $Ind(B_j\,, G_1^p)$ signifies that the parameter B_j is independent of the set G_1^p, whereas $Dep(B_j\,,\, G_1^p)$ states that the parameter B_j is dependent on the set G_1^p.

The number of the primary configurations S_d^p can then be expressed as follows:

$$S_d^P = \begin{cases} 1 + \sum\limits_{\forall B_i \in G_1^p} \left[\eta\left(B_i\right) - 1\right] + \frac{k(k-1)}{2} & , \; Ind(G_1^p) \\ \prod\limits_{\forall B_i \in G_1^p} \eta\left(B_i\right) & , \text{otherwise,} \end{cases} \tag{3.4}$$

where k is the number of parameters in the primary set G_1^p, and $Ind(G_1^p)$ means that all the parameters in the primary set are independent of each other. Looking at the upper expression of Eq. 3.4 which is applied when the primary set G_1^p is independent ($Ind(G_1^p)$), the first term represents the first experiment needed to be conducted which could have any parameter settings. Since the primary set is independent, the second term represents the number of test cases required to be conducted across each parameter in the primary set with subtracting one state, which was conducted in the first experiment, from each parameter. The third term which was derived from the binomial coefficient formula stands for the minimum number of test cases that are required to examine the independency among the parameters. The lower expression basically represents all possible combinations of the parameters in the primary set since they are dependent on each other.

The number of standalone configurations S_d^s related to a standalone parameter B_j is Q_j which can be expressed as:

$$Q_j = \begin{cases} \eta(B_j) - 1 + k & , Ind(B_j\ , G_1^p) \\ [\eta(B_j) - 1] \left[1 + \sum\limits_{\forall B_i \in G_1^p} (\eta(B_i) - 1) \right] & , Ind(G_1^p) \\ [\eta(B_j) - 1] \prod\limits_{\forall B_i \in G_1^p} \eta(B_i) & , Otherwise, \end{cases} \tag{3.5}$$

where $Ind(B_j, G_1^p)$ means the standalone parameter B_j is independent of all primary parameters G_1^p. In the case where B_j is independent of G_1^p in Eq. 3.5, the first and second terms represent the number of experiments required to be conducted across B_j where one of the states of B_j was already conducted in the primary stage. The third term represents the minimum number of experiments required to verify the independency of B_j among all parameters in G_1^p.

In the case where G_1^p is independent, the same number of configurations required in the primary configurations in the case where G_1^p is independent is needed to be conducted for the rest of the states of B_j; in here we don't need to include the term used to examine the independency because we would already know, from the primary stage, that G_1^p is independent. When G_1^p is a dependent set, all possible combinations are required to be considered with subtracting the configurations which were conducted in the primary stage.

The number of all standalone configurations S_d^S can now be expressed as:

$$S_d^S = \sum_{\forall B_j \in G_1^s} Q_j \tag{3.6}$$

Considering the case where only one parameter is in the primary set ($k = 1$) gives the special case shown in [69] (where the primary set contains only one parameter.)

3.1.3 Description of the Methodology

In this section, a *HTC Nexus One* smartphone will be considered as an example to explain the flow chart shown in Fig. 3.1 and to show how to fill up the configuration table shown in Table 3.2.

1. As a first step, all the phone's parameters have to be grouped in a set G^d. The *HTC Nexus One* smartphone has 21 parameters that affect its power consumption [4]. However, parameters in G^d are categorized into three sets, G_0^d, G_1^d and G_2^d. The set G_0^d includes the basic parameters. For the *HTC Nexus One* phone, there are six basic parameters. The set G_1^d includes the active parameters that control the hardware components. the *HTC Nexus One* smartphone has five active parameters ($m_A = 5$). The set G_2^d contains the passive parameters that use, but do not control, the hardware components. There are ten passive parameters in the *HTC Nexus One* phone.

   The parameters in G_1^d are further classified into two subgroups, G_1^p and G_1^s. The primary parameters G_1^p appear to consume a large proportion of energy. The rest of the active parameters are called standalone parameters G_1^s. Techniques for selecting primary parameters G_1^p are beyond the scope of this work. For the *HTC Nexus One* smartphone, only Data Access Mode and Volume are considered in G_1^p; that is $G_1^p = \{B_{31}, B_{34}\}$ and $k = |G_1^p| = 2$.

2. After categorizing the parameters, all settable parameters (G_1^d and G_2^d parameters) should be set to initial values $b_j^0 \Big|_{B_j \in \left(G_1^d \cup G_2^d\right)}$. Passive parameters are kept fixed to those initial values during the whole experiment. Now, we need to initiate the set of configurations $Tc = \phi$. This set will include all the selected configurations.

3. We need to check the dependency among all primary parameters G_1^p. In order to do so, we need to conduct some experiments. Two parameters are said to be independent if their energy costs are additive [4]. The dependency has to be checked for each and every pair of the primary parameters G_1^p. The function $Ind(G_1^p)$ takes the set of the parameters G_1^p and checks the dependency among each and every pair of the parameters in that set. The outcome of this function is either "*True*," if all the parameters are independent of each other or "*False*," otherwise. To check the dependency for

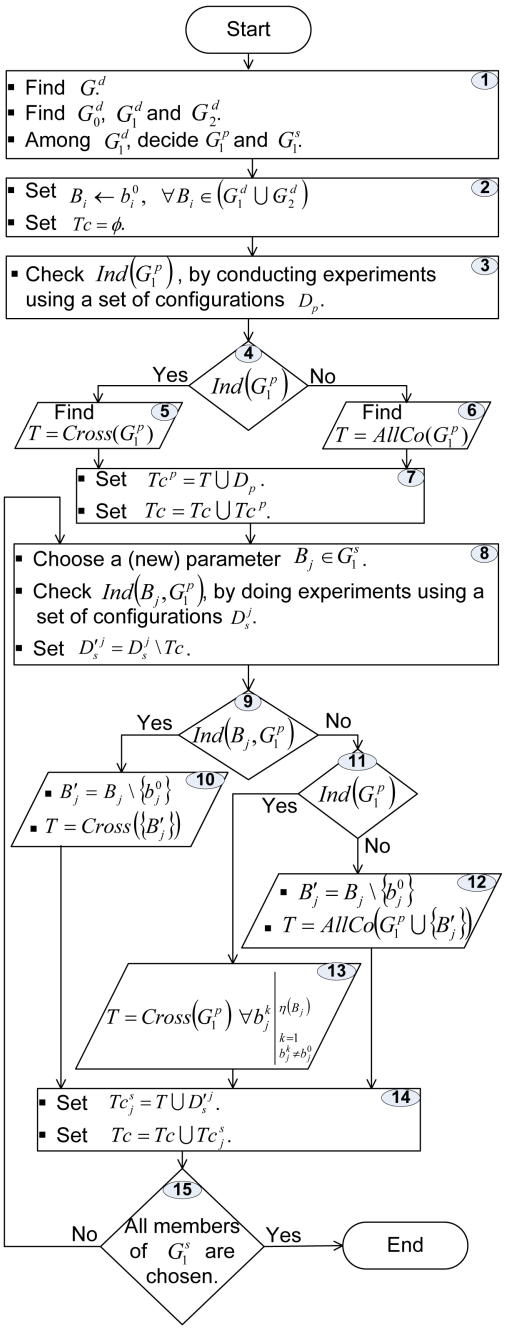


Figure 3.1: Flowchart for filling out the configuration table, (*AllCo* and *Cross* are explained in the text, and T is a local variable).

Table 3.2: Configurations table for *HTC Nexus One* smartphone ($m_A = 5$).

Stage	Configurations	Primary parameters $k = 2$		Stand alone Parameters $m_A - k$		
		Data Access Mode $\eta(B_{34}) = 3$	Volume $\eta(B_{31}) = 16$	GPS $\eta(B_{35}) = 2$	Bluetooth $\eta(B_{33}) = 2$	Brightness $\eta(B_{32}) = 5$
Primary configurations	C_1^t	Off	0	Off	Off	0%
	C_2^t	Off	15	Off	Off	0%
	C_3^t	3G	0	Off	Off	0%
	C_4^t	3G	15	Off	Off	0%
	C_5^t	WiFi	0	Off	Off	0%
	C_6^t	Off	1	Off	Off	0%
	C_7^t	Off	2	Off	Off	0%
	C_8^t	Off	3	Off	Off	0%
	C_9^t	Off	4	Off	Off	0%
	C_{10}^t	Off	5	Off	Off	0%
	C_{11}^t	Off	6	Off	Off	0%
	C_{12}^t	Off	7	Off	Off	0%
	C_{13}^t	Off	8	Off	Off	0%
	C_{14}^t	Off	9	Off	Off	0%
	C_{15}^t	Off	10	Off	Off	0%
	C_{16}^t	Off	11	Off	Off	0%
	C_{17}^t	Off	12	Off	Off	0%
	C_{18}^t	Off	13	Off	Off	0%
	C_{19}^t	Off	14	Off	Off	0%
Stand alone configurations	C_{20}^t	3G	0	On	Off	0%
	C_{21}^t	3G	15	On	Off	0%
	C_{22}^t	WiFi	0	On	Off	0%
	C_{23}^t	3G	0	Off	On	0%
	C_{24}^t	3G	15	Off	On	0%
	C_{25}^t	WiFi	0	Off	On	0%
	C_{26}^t	3G	0	Off	Off	100%
	C_{27}^t	3G	15	Off	Off	100%
	C_{28}^t	WiFi	0	Off	Off	100%
	C_{29}^t	3G	0	Off	Off	25%
	C_{30}^t	3G	0	Off	Off	50%
	C_{31}^t	3G	0	Off	Off	75%

a pair of parameters, at least four configurations are required. However, the same configurations can be used to check the dependency for many pairs. Although being mainly used to check the dependency, those configurations are also used to fill up the configuration table. The set D_p represents the set of configurations used to check the dependency among G_1^p. In our example, the set of configurations D_p consists of the first four configurations in the Table 3.2. That is $D_p = \{C_1^t, C_2^t, C_3^t, C_4^t\}$.

4. We make a decision about how to get the primary configurations depending on the results of the dependency check. If all parameters in G_1^p are independent (that is, $Ind(G_1^p) = true$), we need to go to step (5) to get the primary configurations. Otherwise, we get the primary configurations via step (6). In our example, we assumed that the two primary parameters we have are independent of each other; that is $Ind(G_1^p) = true$.

5. We try to get a set of configurations T representing a subset of the primary configurations Tc^p. Since the primary parameters are independent, we need to choose configurations only across each of the primary parameters [4]. The function $Cross(G_1^p)$ takes the primary parameters G_1^p and returns a set of configurations T by going across each of the primary parameters. In our example, by going across both of the primary parameters, we get $T = \{C_1^t \sim C_3^t, C_5^t \sim C_{19}^t\}$.

6. Since the primary parameters are not independent, we need to consider all possible combinations of the primary parameters in order to get the set of configurations T. The function $AllCo(G_1^p)$ takes the primary parameters and returns a set of configurations T by considering all possible combinations of those parameters.

7. Some of the configurations in the set T were already considered in the set of configurations D_p used to check the dependency among primary parameters. The set of all primary configurations Tc^p can now be determined by the union of T and D_p; that is $Tc^p = T \bigcup D_p$. Then, the set Tc has to be updated by considering the primary configurations Tc^p in it. In our example, the set of the primary configurations $Tc^p = \{C_1^t, C_2^t, ..., C_{19}^t\}$. We can verify the number of primary configurations by applying Eq. 3.4. Now, we update the set Tc; that is $Tc = Tc \bigcup Tc^p = Tc^p = \{C_1^t, C_2^t, ..., C_{19}^t\}$.

8. From now on, we will be dealing with the standalone parameters. We choose one (new) standalone parameter B_j, and then we check the dependency between the standalone parameter B_j and the primary parameters G_1^p. In our example, we chose GPS parameter (B_{35}) [4] as the first standalone parameter. The function $Ind(B_j, G_1^p)$

checks the dependency between the standalone parameter and each of the primary parameters. It returns "*True*" if the standalone parameter is independent of all of the primary parameters. Otherwise, it returns "*False*". We need to conduct some experiments using a set of configurations D_s^j in order to check the dependency. The set D_s^j contains the configurations used to check the dependency between the standalone parameter B_j and each of the primary parameters in G_1^p. We used the configurations $\{C_3^t, C_4^t, C_{20}^t, C_{21}^t\}$ to check the dependency between the GPS parameter and the volume parameter, and the configurations $\{C_3^t, C_5^t, C_{21}^t, C_{22}^t\}$ to check the dependency between the GPS parameter and the Data Access Mode parameter. Thus $D_s^{35} = \{C_3^t, C_4^t, C_5^t, C_{20}^t, C_{21}^t, C_{22}^t\}$. The set $D_s'^j$ is a subset of D_s^j, which represents the configurations in D_s^j that are not yet included in the Tc. That is $D_s'^{35} = D_s^{35} \backslash Tc = \{C_{20}^t, C_{21}^t, C_{22}^t\}$.

9. At this point, a decision about how to get the standalone configurations is made based on the results of the dependency check. If the standalone parameter B_j is independent of all the primary parameters, we go to step (10). Otherwise, we go to step (11). In our example, we assumed that standalone parameters are independent of all primary parameters.

10. At this point, we eliminate the initial state of the standalone parameter B_j from its set of states because the initial state was already considered in the primary stage, and we call the new set of states B_j'. Since $Ind(B_j, G_1^p) = True$, we get standalone configurations related to B_j by going only across B_j'. In our example, only the configuration C_{22}^t is considered ($T = \{C_{22}^t\}$).

11. In this case, we need to look again at the dependency among the primary parameters, which was already checked in step (3). In order to get the standalone configurations, we go to step (13) if the primary parameters are independent, and we go to step (12) otherwise.

12. We eliminate the initial value b_j^0 of the standalone parameter B_j and form a new set of states B_j'. After that, we get the set of configurations T related to B_j by considering all possible combinations between the primary parameters G_1^p and the set $\{B_j'\}$.

13. For this case, we basically consider the primary configurations with the rest of the states of the standalone parameter B_j in order to get the standalone configurations related to B_j. In other words, to get the set of configurations T, we apply the function $Cross(G_1^p)$ with each state of the standalone parameter B_j except the initial state.

14. At this point, we have already got the set of configurations T. We can get the set of standalone configurations Tc_j^s corresponding to the standalone parameter B_j by taking the union between the set T and the set $D_s'^j$. The set Tc is updated by considering the standalone configurations Tc_j^s. In our example, the standalone configurations corresponding to the GPS parameter are $Tc_{35}^s = T \bigcup D_s'^{35} = \{C_{20}^t, C_{21}^t, C_{22}^t\}$, and the set of configurations $Tc = Tc \bigcup Tc_{35}^s = \{C_1^t, C_2^t, ..., C_{22}^t\}$. We can verify that $Q_{35} = |Tc_{35}^s|$ via Eq. 3.5.

15. At this point, we check if there is a standalone parameter which has not been chosen yet. If so, we go back to step (8). Otherwise, we end. In our example, and by assuming that the rest of the standalone parameters (Bluetooth 'B_{33}' and Brightness 'B_{32}') are also independent of G_1^p, we get $Tc_{33}^s = \{C_{23}^t \sim C_{25}^t\}$ and $Tc_{32}^s = \{C_{26}^t \sim C_{31}^t\}$. The configurations $\{20 \sim 31\}$ represent all standalone configurations (see Eq. 3.6).

From the above application for *HTC Nexus One* phone, and referring to Table 3.2, we notice that: $m_A = 5$; $k = 2$; $\eta(B_{34}) = 3$; $\eta(B_{31}) = 16$; $\eta(B_{35}) = 2$; $\eta(B_{33}) = 2$; $\eta(B_{32}) = 5$; $G_1^p = \{B_{34}, B_{31}\}$; $Ind(B_{35}, G_1^p) = True$; $Ind(B_{33}, G_1^p) = True$, and $Ind(B_{32}, G_1^p) = True$. By referring to Eqs. (3.4 $\sim$ 3.6), now we can get $S_d^p = 19$; $Q_{35} = 3$; $Q_{33} = 3$; $Q_{32} = 6$; $S_d^s = 12$, and $S_d = 31$.

Application developers and testers can follow this process to select a reduced set of configurations in order to evaluate the energy performance of their applications. Also, they can use this process to compare applications' behaviors on different devices and platforms.

3.1.4 Experimental Setup

Our approach uses a test bench to measure the power consumption (the output of the test) of the application under test for every test case. As shown in Fig. 3.2, a smartphone is powered by a power supply with a high precision current measurement unit. This power supply is connected to a computer in order to read and store the measurements. During the experiment, we disconnect the battery's power interface and power the smartphone from the power supply; however, the battery communication interface should stay connected to the smartphone. Also the battery has to have enough power during all experiments so that no power-saving state takes place.

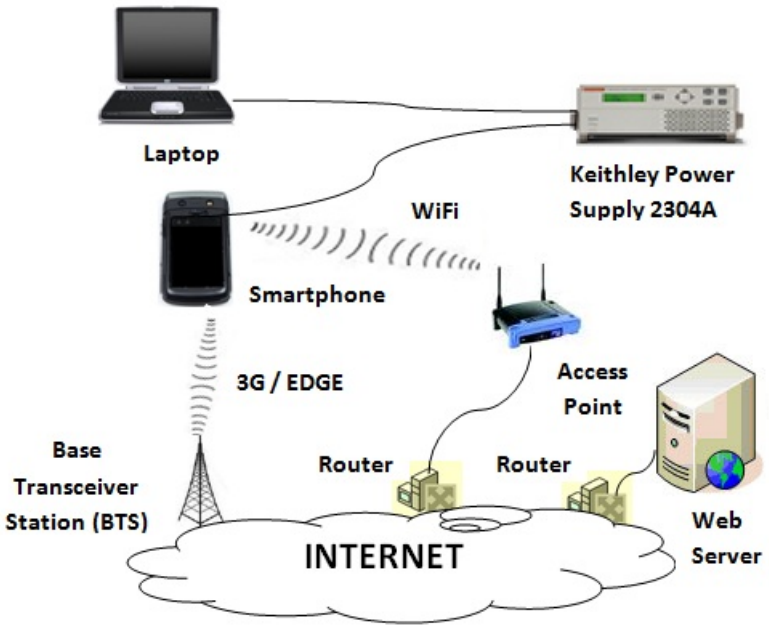


Figure 3.2: Experimental setup.

3.2 Network-Traffic Analysis

In order to have a better picture of the power consumption of network-related applications, a network-traffic analyzer is needed in addition to the power measurement bench introduced earlier in this chapter. Analyzing the traffic generated by an application can provide information about when and for how long the WNIC is transmitting, receiving. idling or sleeping. For this purpose, Wireshark, a network protocol analyzer, is used to sniff the network traffic in the wireless link between the access point and the smartpnone. Figure 3.3 shows where the traffic sniffing is taking place. Wireshark is installed in a laptop running linux kernel. In order to be able to sniff the traffic wirelessly, the WiFi interface of the laptop is switched into monitoring mode. Also, WEP encryption information is given to wireshark so that wireshark is able to decrypt the IEEE802.11 packets.

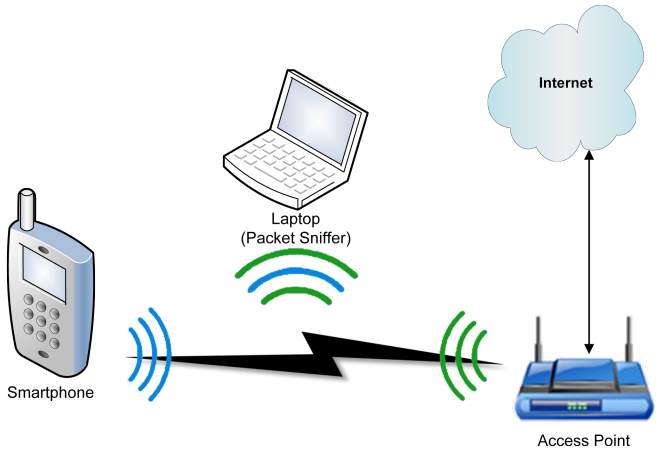


Figure 3.3: Network traffic analysis at the wireless link.

Sniffing the traffic wirelessly helps to track the *Power Flag* in the headers of IEEE 802.11 frames. This flags tells when a station is switching to a sleep mode (PSM). Also, the broadcast frames sent by the access point and the *Null Function* frames can be tracked. In addition to that, the signal straight and noise information in the wireless channel can also be monitored.

Chapter 4

Streaming Strategy

Mobile video streaming applications have become the most popular trends in smartpnone applications. However, these kinds of applications consume lots of energy and thus significantly hurt user experience. In this chapter, we test the energy consumption of delivering multimedia contents to streaming applications and suggest a client-centric streaming technique.

4.1 Motivation

As was shown in chapter 2, there are many different techniques that can be applied for streaming applications in order to enhance the energy cost of multimedia delivery. At the application level, different application developers could adopt different delivery techniques to build their streaming applications. Motivated by this, we have used the testing technique explained in chapter 3 on different smartphone streaming applications and compared their energy consumption. We chose YouTube streaming applications as YouTube is one of the most popular video streaming providers. As of the end of 2011, it was reported that 10% of the Internet traffic in North America was YouTube sessions [39]. First, the impact of different video contents on the energy consumption of a YouTube applications was explored. Five different 5-minute YouTube video contents with the same resolution were played while testing the energy consumption of the YouTube application on *iPhone 3GS* over a WiFi interface. Figure 4.1 shows the power consumptions of streaming 5 different YouTube Videos. As the figure shows, the power cost of delivering the video contents to the client application is almost twice the power cost of decoding and playing them. The YouTube application starts playing the video contents after it has enough frames buffered,

and it then keeps downloading and playing at the same time. At some point, the whole video content is downloaded, and the application continues playing the video from the playback buffer. Also, this figure shows that content variation has very little impact on energy consumption. Different video contents with the same length and the same resolution consume the same battery power.

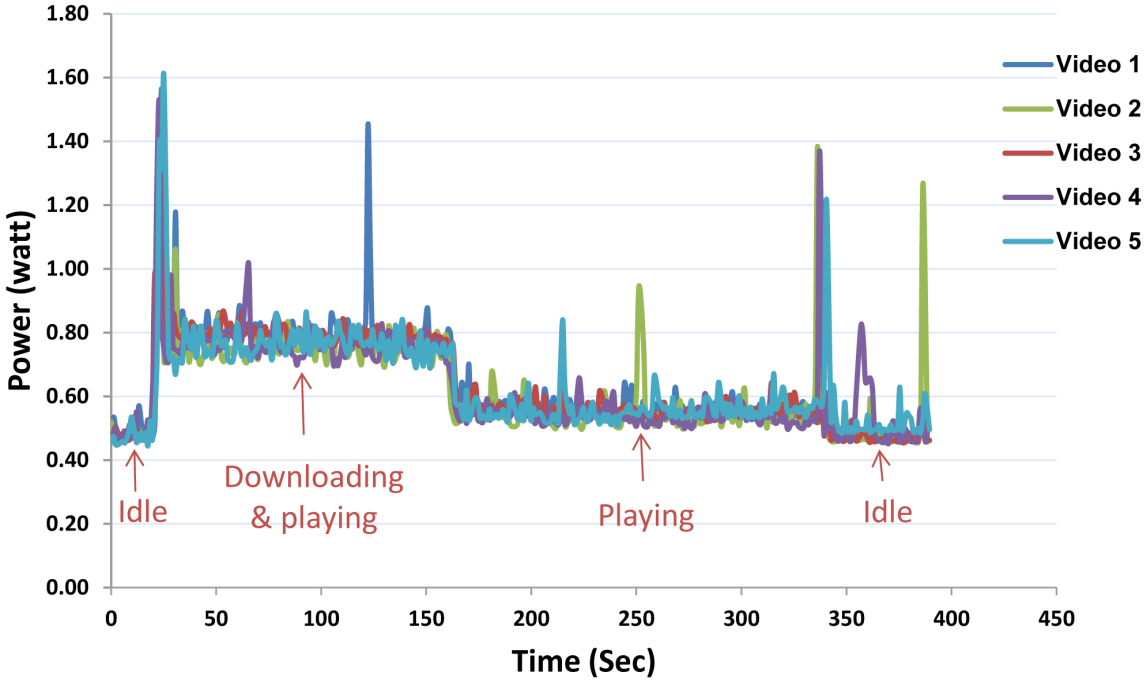


Figure 4.1: Streaming 5 different 5-minute YouTube contents using the YouTube App to an iPhone via WiFi interface.

Having known that the content variation does not have much impact on energy consumption, we compare the energy costs of streaming video content on two applications: namely YouTube App and FoxTube App (both of them are YouTube applications found in Apple App Store.) We streamed the same video content from the same server to the same client (*iPhone 3GS*) using the two YouTube applications mentioned. Figure 4.2 shows the power consumption of streaming a 10 minute video content to the two applications. Even though exactly the same video content was streamed from the same server over the same network to the same mobile device, FoxTube App finishes downloading the whole video contents in about 40 seconds, whereas it takes YouTube App about 283 seconds to finish downloading the whole video. Thus, the YouTube App consumes 13% more energy than the FoxTube App. As is shown in the figure, the power consumption of playing the video

content is the same for both applications. That is to say that the two applications use different techniques to download the video content, which led them to consume different amounts of energy. It is worth mentioning that the two applications give exactly the same user experience for streaming the video content.

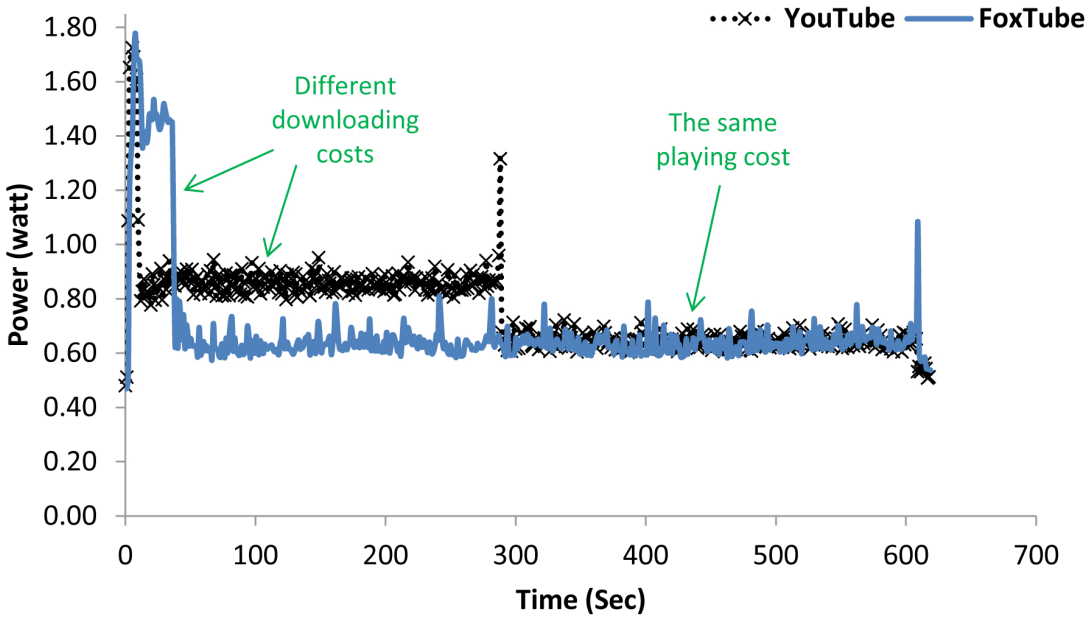


Figure 4.2: Power consumption of streaming a 10-minute Youtube video using two different applications to an iPhone via WiFi interface.

In order to understand the differences in content delivery of the two applications, a traffic analyzer is used as explained in section 3.2. At the transport layer, figure 4.3 shows the Tcptrace of the traffic flows of both applications. This figure confirms that the FoxTube App takes about 40 seconds to finish downloading the whole 10-minute video file, whereas it takes the YouTube App about 284 seconds to download the same file. The sub-figures within figure 4.3 show zoomed-in graphs of both flows. For the FoxTube App, the traffic is continuous, and the server application transmits the file as fast as TCP can deliver. For the YouTube App, however, the traffic flow is discrete, and the server application pushes a burst of about 64KByte to the transport layer and then pauses for some time before pushing the next burst. It takes the TCP about 0.051 seconds to transmit a burst, and then the TCP at the server side has no packets to send for about 0.37 seconds when the server application is pausing. In other words, the YouTube App applies an application-level flow control, whereas the FoxTube App relies on the TCP flow control. This explains why

it takes the YouTube App that long to finish downloading the whole file. However, it does not really explain why the YouTube App consumes 13% more energy than the FoxTube.

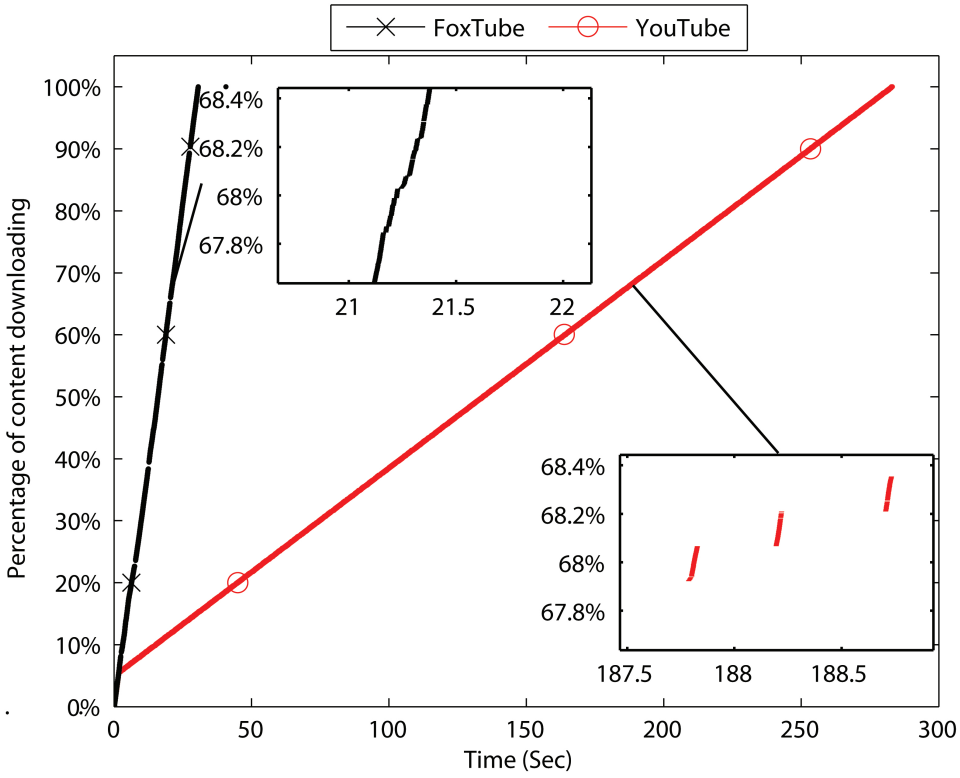


Figure 4.3: Tcptrace of streaming a 10-minute Youtube video using two different applications to an iPhone via the WiFi interface.

In order to understand the differences in the energy consumptions of these two applications, it is worth looking at what is happening at the link layer. The WiFi interface was used for both applications. The iPhone, as most of today's smartphones, uses the PSM-A technique (explained in section 2.2.1, page 10). If the WNIC is in a CAM (Continuous Awake Mode) and there is no traffic (or the traffic is less than a threshold) for a timeout, the WNIC will switch to PSM. Based on our experiments and the results in [43], most of the smartphones' WNICs have a timeout between 30ms to 200ms. For the *iPhone 3GS*, the timeout is 60ms. Because the server application pauses for 370ms after each burst when using the YouTube App, the WNIC of the iPhone stays idle for 60ms (the timeout period) after each and every burst and then switches to the PSM. Tracking the *Power Flag* in the IEEE 802.11 frames shows how long the WNIC is active (CAM), idle (timeout) and asleep (PSM). Figure 4.4 shows the percentages of time the WNIC is in a particular state for both

applications. The figure indicates that the WNIC is idle for 7.38% of the time when the YouTube App is used, whereas it is idle for only 0.07% of the time when the FoxTube App is used. This explains the reason why the YouTube App consumes 13% more energy than the FoxTube App while the same video content is being streamed from the same server for both applications.

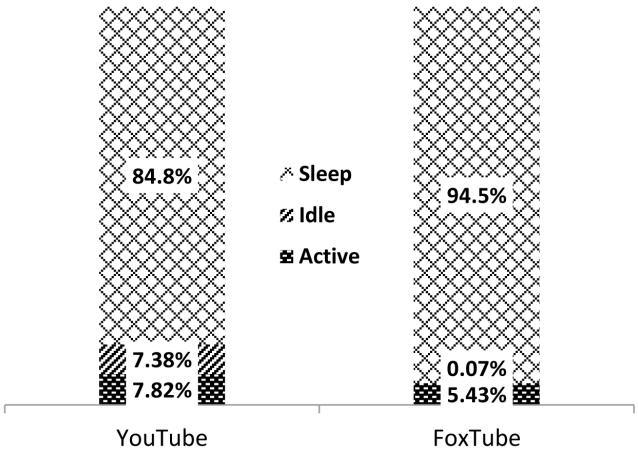


Figure 4.4: The percentage of time for each of the WNIC states while streaming a 10-minute Youtube video using two different applications to an iPhone via WiFi interface.

Both applications streamed the multimedia content from the same YouTube server. The FoxTube App forces YouTube server not to apply application-level flow control by setting the parameter "*ratebypass=yes*" in the URL of the HTTP request, whereas the YouTube App does not set this parameter. It is worth mentioning that decisions made by application developers at the application level directly impact the energy consumption of the underneath protocols in the other layer of the protocol stack.

Motivated by this, we have suggested a client-centric streaming technique. In this technique, the client controls the downloading process, and it saves energy by generating big bursts separated by long time periods. By doing this, the number of transitions between CAM and PSM is reduced, and consequently the amount of time the WNIC is in the idle state is also reduced.

4.2 System Model

In our model, the client controls the download process. Figure 4.5 shows the system model. As shown in the figure, the client application consists of three main components, namely

Downloader, Buffer and Player. The player takes multimedia frames from the buffer at a constant rate and plays them back. The downloader fetches the multimedia frames from the server and puts them in the buffer. The downloader and the buffer play the most important roles in our model since we are mainly interested in content delivery.

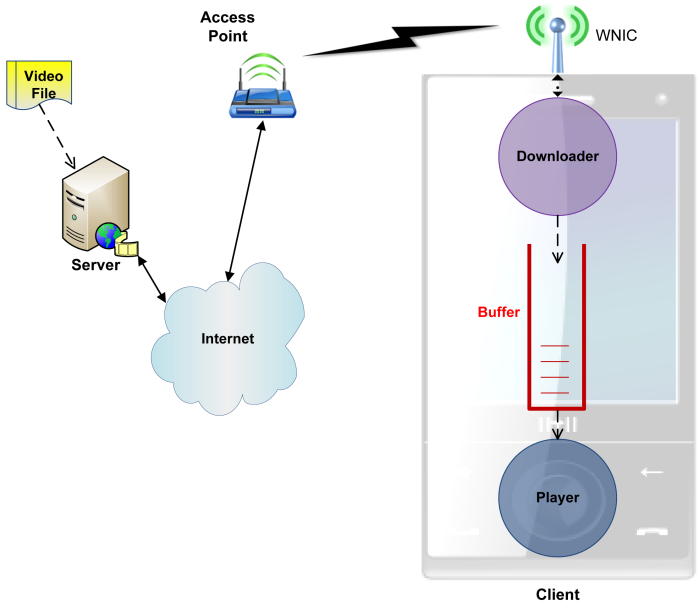
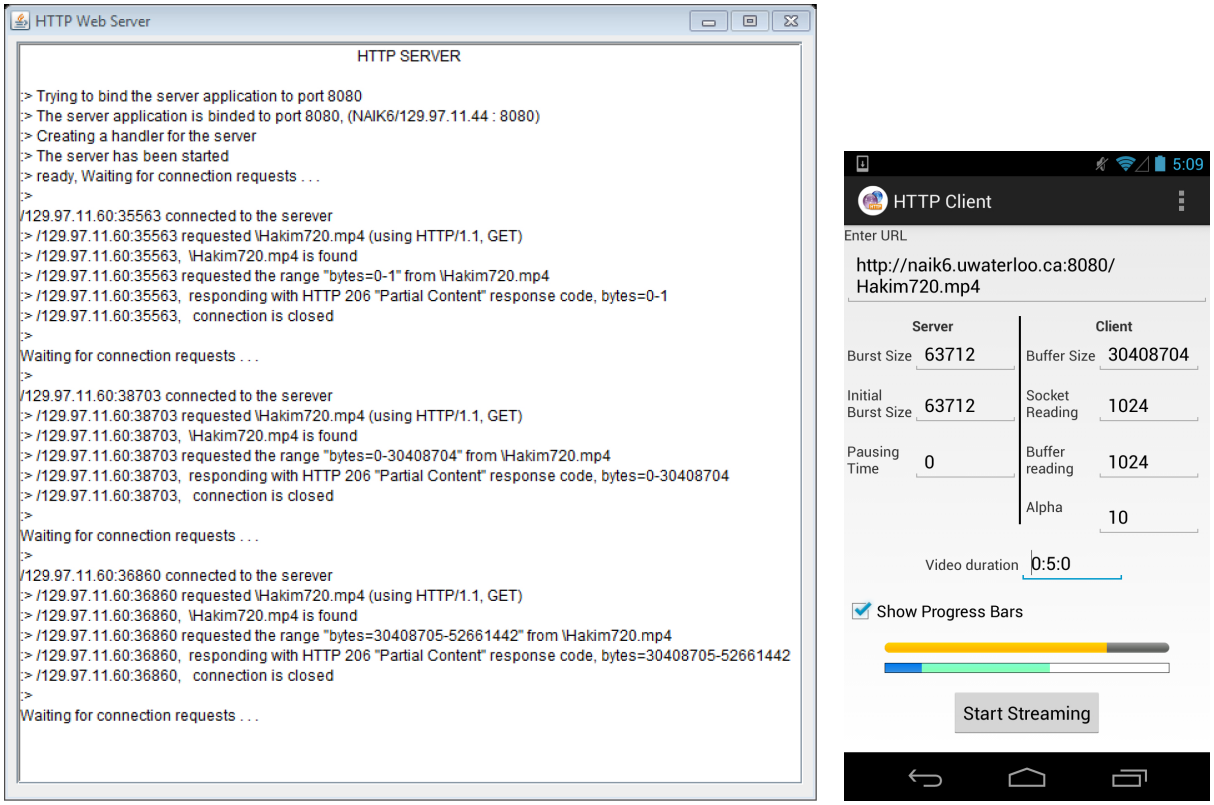


Figure 4.5: System model.

We have implemented an HTTP server using JAVA programming language. The server is deployed in a desktop machine (located in our lab) running *Windows 7*. Also, we have implemented a client-side streaming application using *Android SDK*. The client application is deployed in a *Galaxy Nexus* smartphone running *Android V4.2.1*. The client is connected to a *Cisco Linksys* access point over *IEEE802.11g* interface. All the experiments were conducted using these settings. Figure 4.6 shows the GUIs of the client and the server applications.

The downloader opens a connection with the server and downloads a range of multimedia bytes (using the *Range* field in the header of HTTP *Get* method) that will just fill up the buffer. The Player starts playing back the multimedia content as soon as there are enough frames in the buffer. Once the buffered frames are below a Low Water Mark (LWM), the downloader fetches the next range of bytes that will just fill up the buffer.

The pseudocode shown in Algorithm 1 explains how the downloader works. The process takes *1*) file size, *2*) buffer size, *3*) LWM, and *4*) socket-reading as its input. The file size is



(a) Server GUI. (b) Client GUI.

Figure 4.6: The GUIs for the server and the client's streaming applications.

commonly obtained by sending an HTTP request specifying the *range* field to be the first couple of bytes of the file. The server sends back an HTTP response specifying the *Content-Length* field to be the file size. The output of the downloader, line 5, is the downloaded file. The process starts, at line 6 in the algorithm, by initializing a counter for the number of bytes downloaded so far. Line 8 calculates the range of bytes to be downloaded (the burst size), which directly depends on the buffer size. The function "*occupied_Buffer*" returns the amount of data in the buffer. The function "*fetch_Range*" at line 9 sends an HTTP request specifying the requested range of bytes. The received TCP stream is accessible by the input byte stream "*new_Data*[]." The byte array "*bytes*[]" used to read the input TCP stream is initialized with a length of "*socket_Reading*" bytes at line 10. At line 12, the function "*read_Stream*" reads "*socket_Reading*" bytes form the input stream "*new_Data*" and saves them in the byte array "*bytes*[]." The function "*buffer_Data*" buffers the bytes read from the input stream into the playback buffer. After all the data in the input stream

"*new_Data*[]" is read, the downloader keeps monitoring the state of the playback buffer at lines 16 to 18. The function "*Sleep*" at line 17 causes the current thread to sleep for the specified amount of time, and the function "*time_Needed_To_Playback*" returns the amount of time required to play back the specified amount of data. Once the data in the playback buffer is below the LWM, the downloader goes back to line 8 and starts fetching the next range of bytes.

Algorithm 1: Client-centric streaming downloader

Input:
1: $file\_Size$
2: $buffer\_Size$
3: LWM ▷ Low Water Mark
4: $socket\_Reading$

Output:
5: $Downloaded\ \ file$

Procedure:
6: $downloaded\_Bytes \leftarrow 0$
7: **while** $downloaded\_Bytes < file\_Size$ **do**
8: $\quad range \leftarrow \min\{(buffer\_Size - occupied\_Buffer\,(\,))\,,$
$\quad\quad (file\_Size - downloaded\_Bytes)\}$
9: $\quad new\_Data\,[\,] \leftarrow fetch\_Range(downloaded\_Bytes + 1\,,$
$\quad\quad downloaded\_Bytes + range)$
10: $\quad Initializing \Rightarrow bytes\,[socket\_Reading]$
11: $\quad$ **repeat**
12: $\quad\quad bytes\,[\,] \leftarrow read\_Stream(new\_Data\,[\,]\,, socket\_Reading)$
13: $\quad\quad buffer\_Data(bytes\,[\,])$
14: $\quad$ **until** *new_Data has been fully read*
15: $\quad downloaded\_Bytes \leftarrow downloaded\_Bytes + range$
16: $\quad$ **while** $occupied\_Buffer\,(\,) > LWM$ **do**
17: $\quad\quad Sleep(time\_Needed\_To\_Playback(occupied\_Buffer\,(\,) - LWM))$
18: $\quad$ **end while**
19: **end while**

4.3 Mathematical Model

The energy consumption of the client-centric technique described in 4.2 can be mathematically modeled as follows:

$$E_{downloading} = \frac{F_S}{P_S} E_{packet} + NoC \times E_C \tag{4.1}$$

where
$E_{downloading}$ is the energy consumption of downloading a given multimedia file with a given buffer;
F_S is the size of the multimedia file to be downloaded;
P_S is the the packet size without headers overhead (equals to the MSS "Maximum Segment Size" of TCP). For most of the experiments we have conducted, $P_S = 1448\ bytes$;
E_{packet} is the Energy consumption of downloading one packet;
E_C is the the energy required for establishing and tearing down one connection;
NoC is the number of connections required to download the whole multimedia file for a given buffer size. NoC is expressed as follows:

$$NoC = \begin{cases} \left\lceil \frac{F_S - B_S}{B_S(1-\alpha)} + 1 \right\rceil & ;\ F_S \geq B_S \\ 1 & ;\ Otherwise \end{cases} \tag{4.2}$$

where
B_S is the buffer size, and
α is the minimum percentage of data in the buffer; that is $\alpha = \frac{LWM}{B_S}$, where $\alpha \in (0, 1)$.

4.3.1 Energy Required for Establishing and Terminating a Connection (E_C)

The energy required to establish and tear down one connection is measured experimentally. E_C is an overhead that is required each time a new range of data is fetched. This overhead includes the following:

1. *TCP connection establishment:* When a range of data is required, an HTTP request is needed to be sent to the server. A TCP connection is required to send this request to the server and to receive the corresponding response from the server. In order to

establish a TCP connection, a three-way handshake packets have to be exchanged between the client and the server.

2. *HTTP request/response:* Each time a new range is required, the client streaming application has to tell the server application what is the range of data it needs . In order to do that, the client sends an HTTP request with a Range Field indicating the required range of data. The server sends an HTTP response confirming the range and the encryption used. Then the server application sends the required range of data to the client application.

3. *TCP connection termination:* After the required range of data is received, the TCP connection is probably not needed in the short run. So, the client and the server terminate the connection. A typical TCP tear-down requires four-way handshake packets to be exchanged between a client and a server.

4. *WNIC idle period:* After a TCP connection is terminated, the WNIC at the client side stays in CAM mode waiting for the timeout to elapse before switching to PSM mode (explained in section 4.1). This timeout is hardware-specific, and different smartphones' network adapters have different timeout values. Table 4.1 shows the WNIC idle timeout, which were determined experimentally, for some smartphones. Also, whenever a WNIC changes mode, it has to notify the access point by sending special packets (called *Null Function*) and setting the *Power Flag* accordingly.

Phone	WNIC idle timeout (ms)
Nokia E71	130
iPhone 3GS	60
iPhone 4	75
Galaxy Nexus	200

Table 4.1: WNIC timeouts of different smartphones.

5. *WNIC mode switching:* After the idle timeout elapses, the WNIC switches its mode into PSM. Similarly, once the traffic rate at the WNIC exceeds a threshold, the WNIC switches its mode into CAM. The process of changing modes at the hardware level costs some time and energy.

These costs all together accumulate for the coefficient E_C. We have measured the E_C for *Galaxy Nexus* smartphone. This measurement was done in two stages:

1. Over one TCP connection, the client successively establishes 1000 HTTP persistent connections. The client sends an HTTP request to the server asking for one byte of data, and the server responds with an HTTP response. Figure 4.7 shows the power consumption of opening 1000 HTTP connections using a *Galaxy Nexus* smartphone. From this experiment, the energy consumption of an HTTP request/response pair can be measured.

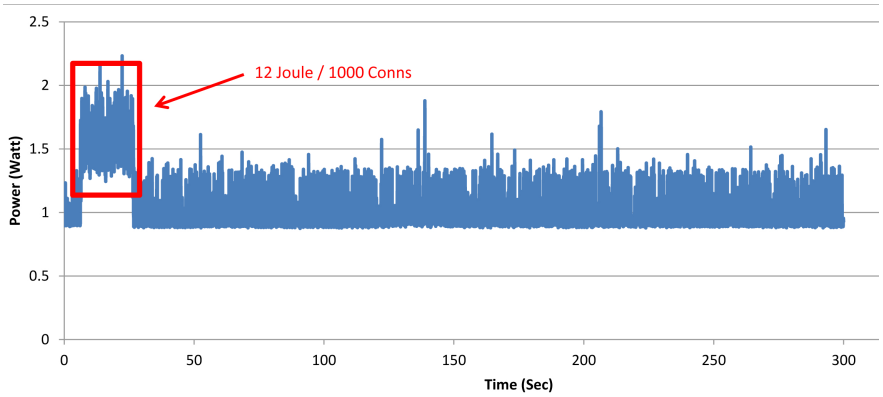


Figure 4.7: Power consumption of openning 1000 HTTP persistent connections in *Galaxy Nexus* smartphone.

2. In this stage, 1000 TCP connections are successively established with a 0.21 second pause time in between. Because the WNIC idle timeout for *Galaxy Nexus* (as indicated in table 4.1) is 0.2 second, the pause between the TCP connections was chosen to be 0.21 second. From this experiment, the energy consumption of opening a TCP connection, terminating a TCP connection, waiting for the WNIC timeout (idle period), and switching a WNIC mode can be measured. Figure 4.8 shows the power consumption of openning 1000 TCP connections with a 0.21 second pause in between for a *Galaxy Nexus* smartphone.

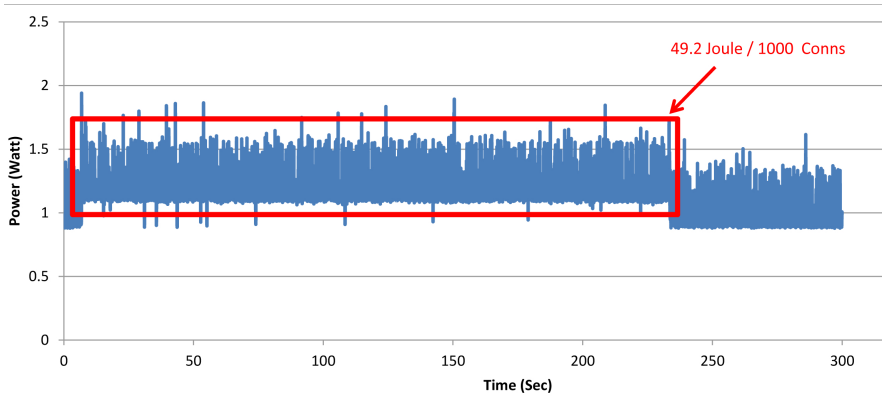


Figure 4.8: Power consumption of openning 1000 TCP connections with 0.21 second pausing time for *Galaxy Nexus* smartphone.

For the *Galaxy Nexus* smartphone, combining the energy costs from the two stages gives:

$$E_C = \frac{12 + 49.2}{1000} = 0.0612 \ \ \text{Joule}$$

4.3.2 Energy Required to Download one Packet (E_{Packet})

The server application delivers the multimedia stream to the TCP socket at the server side; then the TCP protocol at the server side packetizes the stream and sends it to the client-side TCP protocol which will deliver it to the client-side streaming application. The energy cost of downloading one packet (E_{Packet}) consists of the energy cost of receiving the packet and the energy cost of sending the corresponding acknowledgment. This energy cost is determined experimentally by downloading a range of bytes that will make up a certain number of full-sized packets. Thus, the energy cost of downloading a packet is nothing but

$$E_{Packet} = \frac{Energy\ of\ downloading\ a\ range\ of\ bytes - E_C}{Number\ downloaded\ packets}.$$

By using a *Galaxy Nexus* smartphone and our test network, figure 4.9 shows the energy cost of downloading a range of bytes packetized into $10,000$ full-sized packets (1448 byte each). The energy cost of downloading one packet for *Galaxy Nexus* smartphone (E_{Packet}) is about 0.00067053 Joule.

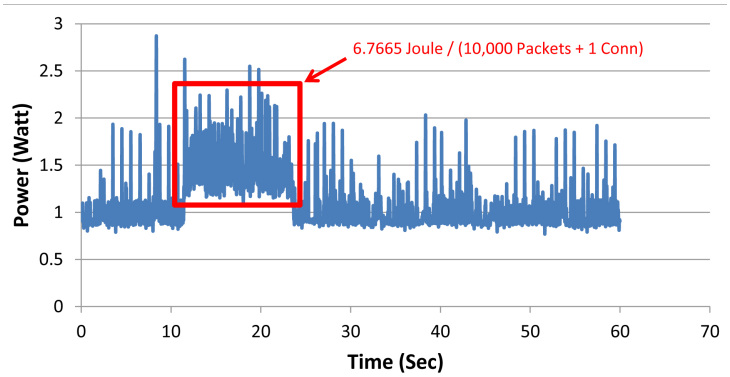


Figure 4.9: Power consumption of downloading 10, 000 full-sized packets for *Galaxy Nexus*.

4.4 System Parameters

There are some parameters associated with this client-centric streaming technique. Some of these parameters are associated with the server, and other parameters are associated with the client. These parameters are as follows:

1. Buffer size (at the client).
2. Low water mark (LWM) for the buffer (at the client).
3. The amount of data read from the client's TCP socket at a time, termed as *socket-reading* (at the client).
4. The amount of data read from the playback buffer at a time, termed as *buffer reading* (at the client)
5. The burst size sent by the server (at the server).
6. The pausing time (at the server).

As far as this work is concerned, only client-side parameters are considered. Buffer size is one of the most important parameters at the client. The Low Water Mark (LWM) is the minimum amount of data in the buffer. During the playback time, the data in the buffer should not be less than the LWM in order to keep continuous playback. The amount of data the downloader reads from the TCP socket at a time and puts in the playback buffer is also another parameter, termed *Socket-reading*. The effects of each of the client side parameters (namely buffer size, LWM, and socket-reading) on the energy consumption of the proposed downloader are studied next.

4.4.1 Buffer Size

The playback buffer size plays an important role in streaming applications. By using an appropriate buffer size, the delays or network throughput fluctuation can be absorbed. Thus, it will allow the client to continue playing back the content during short-term lapses in network bandwidth. However, the main use of the buffer here is not only to compensate for bandwidth variations, but to save energy by allowing the WNIC to sleep for long periods of time while the media content is played back from the buffer.

In order to understand the effect of buffer size on energy consumption, an experiment is needed. The parameter settings used in this experiment are shown in Table 4.2. The test cases chosen for this experiment include changing the buffer size from $61,440$ bytes to $1,026,048$ bytes with steps of $8,192$ bytes. All phone parameters and other application parameters are kept fixed during the experiment. For consistency, the experiment was repeated 4 times. Figure 4.10 shows the measured and the expected changes in energy consumption with respect to the changes in the buffer size. The expected curve was determined using the mathematical model developed in section 4.3, whereas the measured results were observed experimentally using the system model explained in section 4.2. From the figure, the results obtained from the mathematical model match the experimental results. As expected, by increasing the buffer size, the number of required connections NoC is reduced, and consequently the energy consumption of streaming the multimedia file is also reduced. The buffer size is bounded below by the minimum buffer size required to compensate for network variations (which is out of the scope of this work). In the best interest of the user, the buffer size should be greater than or equal to the file size. However, most operating systems limit the amount of memory an application can use. For the *Galaxy Nexus* smartphone running *Android V4.2.1*, our streaming application cannot allocate a buffer size greater than 50 MB.

Parameter	Value
File Size (F_S)	33243196 Byte
Packet Size (P_S)	1448 Byte
LWM (α)	0.1
Video Duration	$5:10$ $m:s$
Socket-Reading	1024 Byte

Table 4.2: Parameter settings for the experiment of changing buffer size.

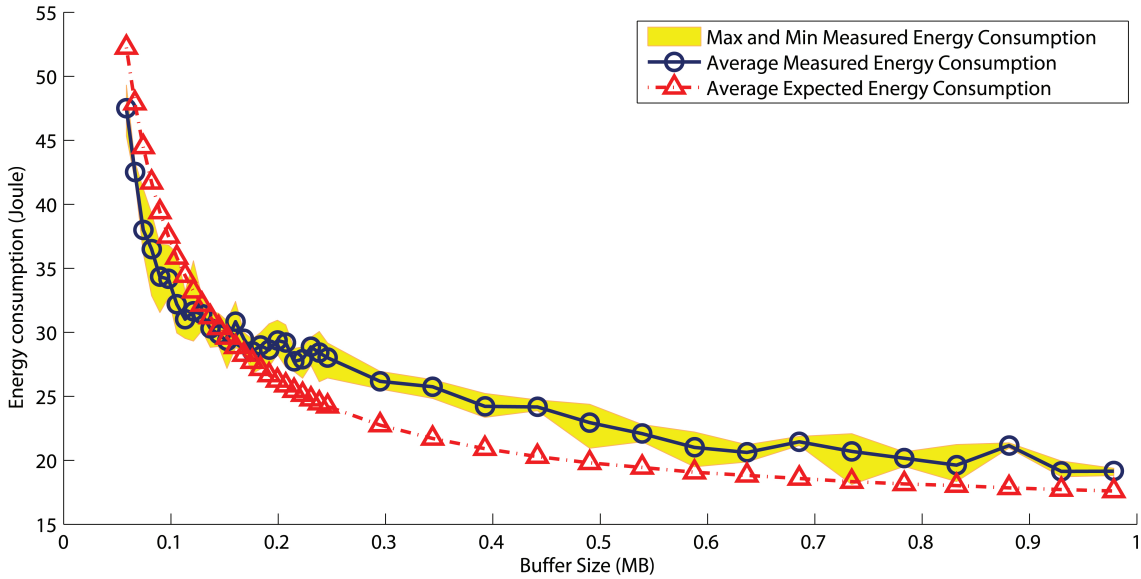


Figure 4.10: Energy consumption of streaming a 31.7 MB, 5 minute video file with different buffer sizes for *Galaxy Nexus* smartphone.

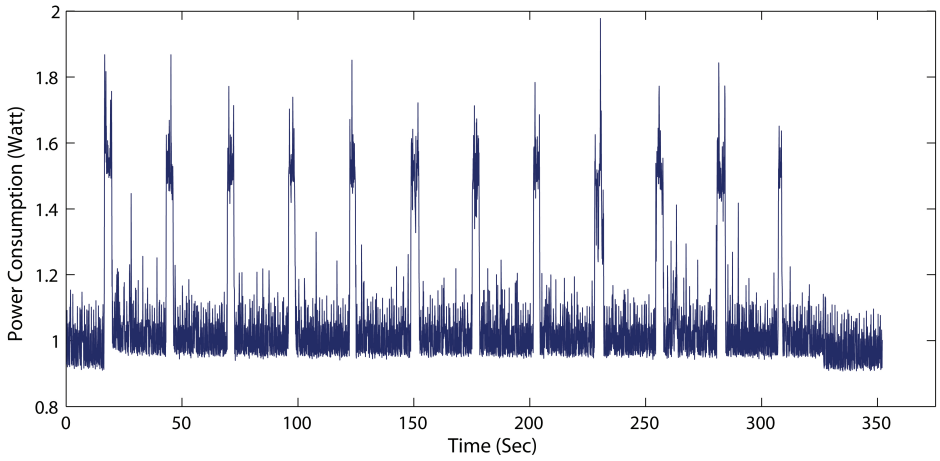


Figure 4.11: Power consumption of streaming a 31.7 MB, 5 minute video file with a 3 MB buffer.

Figure 4.11 shows the power consumption of streaming $31.7 - MB$, $5 - minute$ video content using $3 - MB$ buffer and $LWM = 10\%$. For this experiment, and according to eq. 4.2, the client had to send 12 requests to the server to fully download the video content. The spikes in the figure represent these requests. By having a bigger buffer size, the number

of the spikes is reduced, and hence the overhead cost of establishing new connections (E_C) is reduced. Moreover, the slow-start phase of the TCP protocol leads to an increase in the total downloading time.

4.4.2 Low Water Mark

The LWM is another parameter that affects the number of required connections (eq. 4.2.) To test the effect of the LWM on the energy consumption of a streaming application, an experiment needs to be conducted. Table 4.3 shows the parameter settings for this experiment.

Parameter	Value
File Size (F_S)	53935553 Byte
Packet Size (P_S)	1448 Byte
Buffer size (B_S)	1048576 Byte
Video Duration	$5:00$ $m:s$
Socket-Reading	1024 Byte

Table 4.3: Parameter settings for the experiment of changing LWM.

The parameter α represents the LWM such that $\alpha = \frac{LWM}{B_S}$. The test cases chosen for this experiment include changing the parameter α from 0.05 to 0.95 with 0.05 increment. All phone parameters and other application parameters are kept fixed during the experiment. For consistency, the experiment was repeated 5 times. Figure 4.12 shows the experimental results as well as the expected results. The expected results obtained from the mathematical model match the average measured results. Since the number of connections increases by increasing α, the energy consumption also increases. The smaller the LWM is, the more energy savings. However, the LWM should be big enough to compensate for network variations. The minimum value of the LWM needed to compensate for jitter is out of the scope of this work. Interested readers can find more information in [55] and [56].

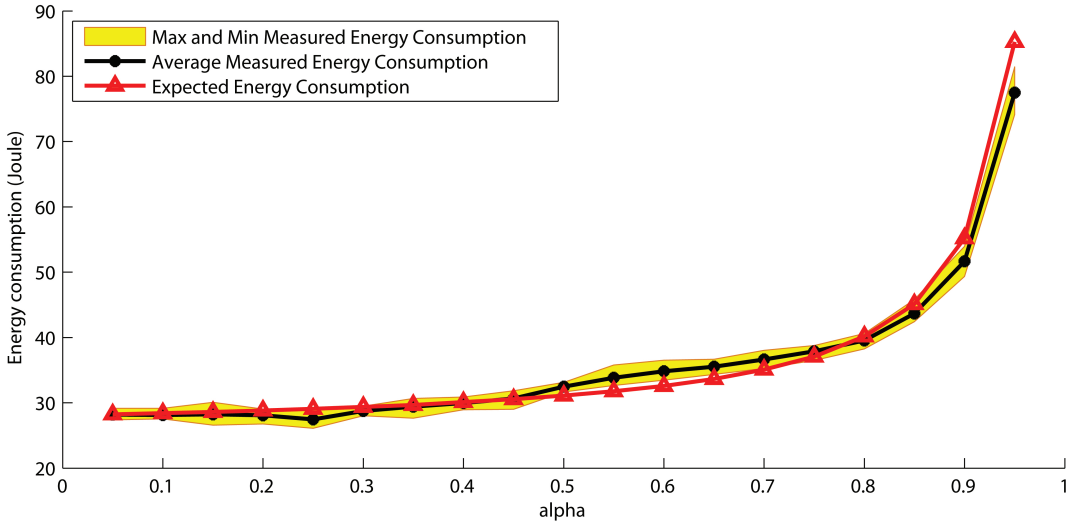


Figure 4.12: The energy consumption of downloading 51 MB, 5 minute video content with different values of α.

4.4.3 Socket-reading

The socket is the interface between the application layer and the transport layer. The term "Socket-reading" means the amount of data an application reads from its input-stream socket at a time. Figure 4.13 shows the interplay of TCP sockets and application processes via buffers. When the server application wants to send content to the client application, it pushes this content to the TCP send buffer via a socket interface. The TCP transmits the content if the congestion and the receive windows allow. The client application reads the content from the TCP receive buffer via the socket interface and stores it into the playback buffer. Socket-reading (the amount of data read from the TCP socket at the client side) is a parameter that can be manipulated by the application developers.

In order to check the impact of the socket-reading parameter on the energy consumption, an experiment was conducted. A 50 MB video file was streamed from the server to the client. The test cases chosen for this experiment include changing the socket-reading parameter from 1 byte to 100 bytes with steps of 1 byte. All other parameters are kept fixed during the experiment. For consistency, the experiment was repeated 4 times. Figure 4.14 shows the results. Surprisingly, the energy consumption of downloading the content when the socket-reading is 1 byte is 150% more than the energy consumption of downloading the content when the socket-reading size is 100 bytes.

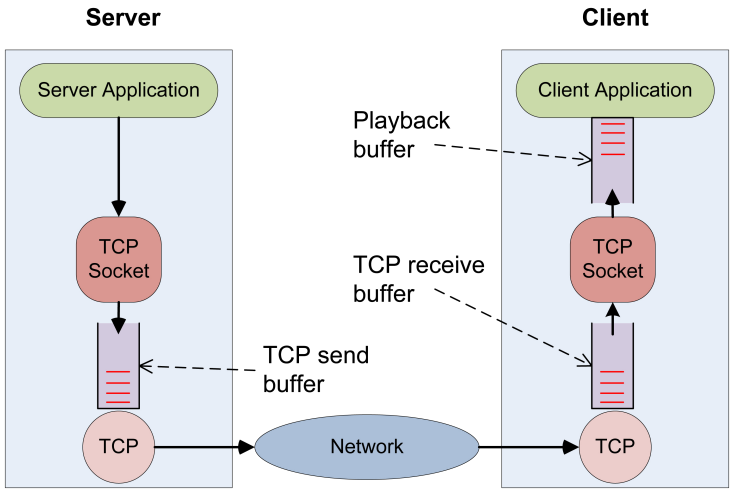


Figure 4.13: Interplay of TCP sockets and applications via buffers.

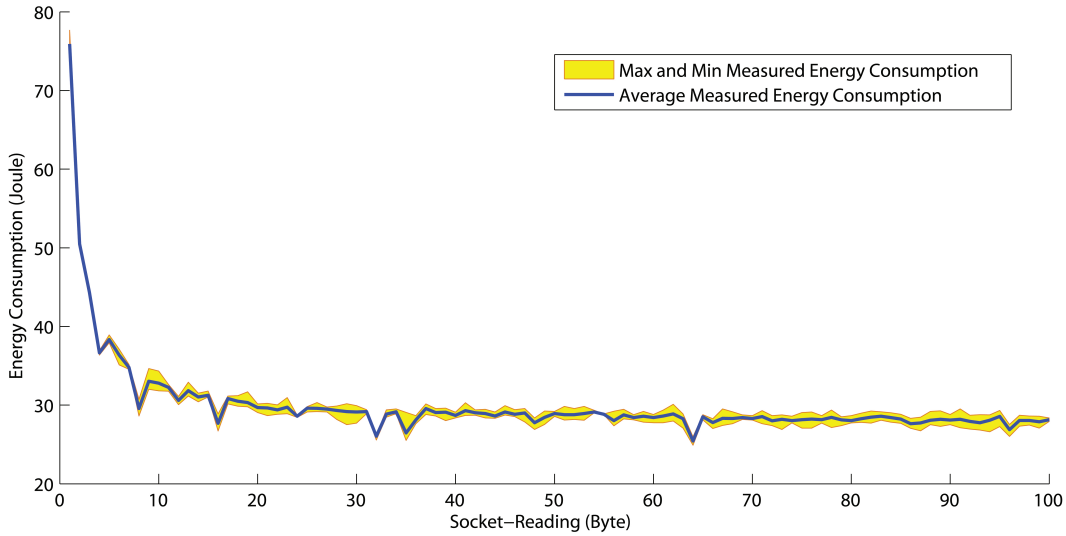


Figure 4.14: The Energy consumption of downloading 50 MB video content for different values of socket-reading parameter.

Figure 4.15 shows the power traces of streaming a 50 MB file with two socket-reading values (1 byte and 100 bytes). The time needed to fully download the file in the case of 1 byte socket-reading is twice as much as that in the case of 100-byte socket-reading. This explains why the energy consumption in the case of small values of the socket-reading parameter is much higher than in the case of large values of the socket-reading parameter.

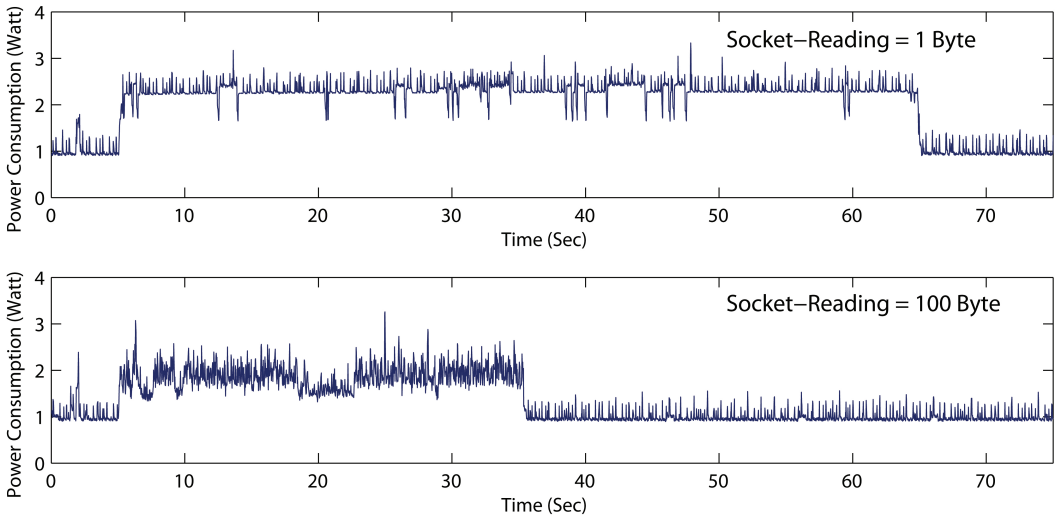


Figure 4.15: The Power consumption of downloading 50 MB video content for two values of socket-reading (1-Byte & 100-Byte).

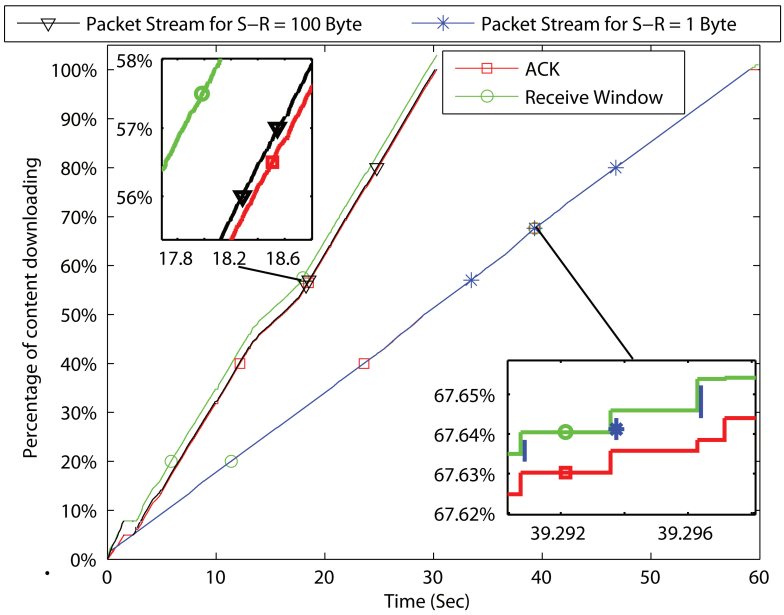


Figure 4.16: TCP-traces of streaming 50 MB video content for two values of Socket-Reading (S-R).

Looking at the TCP layer, Figure 4.16 shows the TCP-traces of streaming the 50 MB

video file in two cases: i) 1 byte socket-reading, ii) 100-byte socket-reading. The streaming application takes about 60 seconds to download the content file in the case of 1 byte socket-reading, whereas it takes about half of that time to download the same content in the case of 100-byte socket-reading. When the streaming application reads 1 byte at a time from the TCP receive buffer (*i.e.* 1 byte socket-reading), the TCP receive buffer at the client side gets full, and consequently the receive windows advertised by TCP at the client side get small. Because of the TCP flow control mechanism and Silly-Window-Syndrome avoidance algorithms, these small advertised receive windows throttle the server and slow down the packet stream. This causes the client device to keep its WNIC active for relatively long time. In the case of the 100-byte socket-reading, however, the server is not constrained by the client receive window as shown in the figure. That means the server sends the packet stream as fast as the network can deliver.

4.5 Discussion

The main idea of this streaming technique is to reduce the amount of time the WNIC is idle. Resulting from using the PSM-A technique in WiFi interface of most of today's smartphones, each time the WiFi interface is to be switched from CAM to PSM, it stays idle for a timeout. The duration of the timeout period is hardware-specific. Based on our observations of some of today's smartphones, it ranges between $30ms$ and $200ms$. Reducing the number of transitions between CAM and PSM will reduce the overall idle time and, thus, reduce the energy consumption.

The buffer size is one of the important parameters that affect the energy consumption. Having a bigger playback buffer at the application level allows to fetch large bursts of data. That reduces the number of transitions between CAM and PSM and saves energy. The LWM is another parameter in streaming applications. It also affects the required number of transitions between CAM and PSM. However, LWM has to be big enough to compensate for network variations and small enough to allow more energy savings.

Socket-reading is another very important parameter. Unlike buffer size and LWM, which affect the idle time of the WNIC, socket-reading affects active time of the WNIC. When socket-reading parameter is small, the streaming application reads slowly from the TCP socket, and, thus, the advertised receive window of the TCP at the client-side will be small. This throttles the available bandwidth and slows down the flow. Eventually, the time needed to download a burst increases, and the WNIC stays active for longer period of time.

Chapter 5

Conclusion

Smartphones are getting increasingly more popular among all users. They have computer-like capabilities and provide sophisticated services to the users. Despite the enormous growth in smartphones ubiquity and their applications and services, their utility has been severely limited by their battery lifetime. Since the growth of the battery is not keeping up with the rapid evolution of other smartphone components, it is very important to conserve smartphones' energy. The hardware components are the actual energy consumer. Software applications, however, use hardware resources through the operating system, which in turn consume battery power when they are in use. Thus, being at the application level, software applications play the most important role in the energy consumption of smartphones.

Owing to the limited battery life, smartphone applications should be energy efficient. Application developers need a tool that helps them make well-functioning applications with minimum power consumption. Alongside with functional testing, energy performance testing is sorely essential. Chapter 3 explains the proposed, generalized methodology of our energy performance testing. Application developers and testers can follow this process to select test configurations and conduct experiments in order to evaluate the energy performance of their applications during the development process. A detailed flow chart was used to explain the process steps, and a specific smartphone, namely *HTC Nexus One*, was used as an example to illustrate the process. We have derived general mathematical expressions representing the number of required configurations. We showed an energy evaluation test bench. It is crucial to standardize a test bench in order to be able to compare test results.

Among all smartphone applications, streaming applications are known to be energy-hungry. On the other hand, they are gaining more and more popularity. From mobile TV to streaming websites, users enjoy streaming multimedia content on their smartphones.

However, user perception is hurt by the limited battery lifetime. It is essential that mobile video streaming applications not only provide a good viewing experience but also avoid excessive energy consumption. Streaming applications consume energy in three stages during a downloading session: *i*) content delivery; *ii*) content decoding; *iii*) content playing. It has been shown that the cost of content delivery is the dominant one among all other costs in streaming applications. The cost of downloading the multimedia content is about treble the cost of decoding it when using the WiFi interface.

In chapter 4, using our proposed test bench, we have evaluated the energy consumption of some popular streaming applications. Among different streaming applications, we found that there is not much different in the decoding and playing costs of streaming a video content. However, application developers adopt different downloading strategies, which cause the applications to consume different amounts of energy while downloading. Many delivery techniques for streaming applications are proposed, which were reviewed in chapter 2. Most of the techniques focus on shaping the stream by making bursts out of constant or variable bit rate traffic.

We have developed a model of an energy-efficient client-centric data downloader for streaming applications. The client controls the downloading mechanism. It saves energy by downloading long bursts into the playback buffer, which reduces the time the WNIC is idle. We have also given a mathematical model representing the energy consumption for our downloader. The effect of each of the downloader's parameters on the energy consumption was studied, and the mathematical model was validated by means of experiments. We have shown that, by increasing the playback buffer of the application, the idle time of the WNIC at the link level decreases, which leads to less energy consumption. Similarly, by decreasing the low water mark (LWM) value, the idle time of the WNIC at the link level decreases. Also, by increasing the socket-reading value, the cost of downloading a packet decreases significantly, which leads to significant energy savings.

There is no panacea for power saving in multimedia streaming on smartphones, and power consumption of streaming applications is still far from optimum. As future work, the server side parameters of streaming applications will be studied. Since some users tend not to watch the whole content while others tend to, user profile should be considered in the model.

References

[1] IEEE standard for information technology - telecommunications and information exchange between systems - local and metropolitan area networks - specific requirements part 11: Wireless lan medium access control (MAC) and physical layer (PHY) specifications amendment 8: Medium access control (MAC) quality of service enhancements. *IEEE Std 802.11e-2005 (Amendment to IEEE Std 802.11, 1999 Edition (Reaff 2003))*, pages 1 –189, 2005.

[2] IEEE standard for information technology– local and metropolitan area networks– specific requirements– part 11: Wireless LAN medium access control (MAC)and physical layer (PHY) specifications amendment 5: Enhancements for higher throughput. *IEEE Std 802.11n-2009 (Amendment to IEEE Std 802.11-2007 as amended by IEEE Std 802.11k-2008, IEEE Std 802.11r-2008, IEEE Std 802.11y-2008, and IEEE Std 802.11w-2009)*, pages 1 –565, 2009.

[3] Ieee standard for information technology– local and metropolitan area networks– specific requirements– part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications amendment 8: Ieee 802.11 wireless network management. *IEEE Std 802.11v-2011 (Amendment to IEEE Std 802.11-2007 as amended by IEEE Std 802.11k-2008, IEEE Std 802.11r-2008, IEEE Std 802.11y-2008, IEEE Std 802.11w-2009, IEEE Std 802.11n-2009, IEEE Std 802.11p-2010, and IEEE Std 802.11z-2010)*, pages 1 –433, 2011.

[4] A. Abogharaf, R. Palit, K. Naik, and A. Singh. A methodology for energy performance testing of smartphone applications. In *Automation of Software Test (AST), 2012 7th International Workshop on Automation of Software Test*, pages 110–116, 2012.

[5] A. Acquaviva, T. Simunic, V. Deolalikar, and S. Roy. Server controlled power management for wireless portable devices. *HP-Labs, HPL-2003*, 82, 2003.

[6] Andrea Acquaviva, Emanuele Lattanzi, and Alessandro Bogliolo. Design and simulation of power-aware scheduling strategies of streaming data in wireless lans. In *Proceedings of the 7th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, MSWiM '04, pages 39–46, New York, NY, USA, 2004. ACM.

[7] Andrea Acquaviva, Tajana Simunic, Vinay Deolalikar, and Sumit Roy. Remote power control of wireless network interfaces. In JorgeJuan Chico and Enrico Macii, editors, *Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation*, volume 2799 of *Lecture Notes in Computer Science*, pages 369–378. Springer Berlin Heidelberg, 2003.

[8] J. Adams and G-M. Muntean. Adaptive-buffer power save mechanism for mobile multimedia streaming. In *Communications, 2007. ICC '07. IEEE International Conference on*, pages 4548 –4553, june 2007.

[9] J. Adams and G.-M. Muntean. Power save adaptation algorithm for multimedia streaming to mobile devices. In *Portable Information Devices, 2007. PORTABLE07. IEEE International Conference on*, pages 1 –5, may 2007.

[10] Manish Anand, Edmund B. Nightingale, and Jason Flinn. Ghosts in the machine: interfaces for better power management. In *Proceedings of the 2nd international conference on Mobile systems, applications, and services*, MobiSys '04, pages 23–35, New York, NY, USA, 2004. ACM.

[11] Manish Anand, Edmund B. Nightingale, and Jason Flinn. Self-tuning wireless network power management. *Wirel. Netw.*, 11(4):451–469, July 2005.

[12] G. Anastasi, M. Conti, E. Gregori, and A. Passarella. Balancing energy saving and qos in the mobile internet: an application-independent approach. In *System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on*, page 10 pp., jan. 2003.

[13] G. Anastasi, M. Conti, E. Gregori, and A. Passarella. 802.11 power-saving mode for mobile computing in wi-fi hotspots: limitations, enhancements and open issues. *Wirel. Netw.*, 14(6):745–768, December 2008.

[14] G. Anastasi, M. Conti, E. Gregori, A. Passarella, and L. Pelusi. An energy-efficient protocol for multimedia streaming in a mobile environment. *International Journal of Pervasive Computing and Communications*, 1(4):301–312, 2005.

[15] G. Anastasi, M. Conti, and W. Lapenna. Power saving policies for wireless access to tcp/ip networks. In *Proceedings of the 8-th IFIP Workshop on Performance Modelling and Evaluation of ATM and IP Networks (IFIP ATM&IP2000), Ilkley (UK)*. Citeseer, 2000.

[16] G. Anastasi, A. Passarella, M. Conti, E. Gregori, and L. Pelusi. A power-aware multimedia streaming protocol for mobile users. In *Pervasive Services, 2005. ICPS '05. Proceedings. International Conference on*, pages 371 – 380, july 2005.

[17] C. Andrews, V. Vitaliev, M. Harris, C. Edwards, J. Hayes, and M. Venables. Predictions for 2013 across our seven sectrors [looking forward 2013 predictions]. *Engineering Technology*, 7(12):29–36, 2013.

[18] A.S.M. Arif, O. Ghazali, and S. Hassan. A survey on buffer and rate adaptation optimization in tcp-based streaming media studies. *Executive Development*, 21:22, 2008.

[19] R. Arya, R. Palit, and K. Naik. A methodology for selecting experiments to measure energy costs in smartphones. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2011 7th International*, pages 2087 –2092, july 2011.

[20] Susmit Bagchi. A fuzzy algorithm for dynamically adaptive multimedia streaming. *ACM Trans. Multimedia Comput. Commun. Appl.*, 7(2):11:1–11:26, March 2011.

[21] D. Bertozzi, L. Benini, and B. Ricco. Power aware network interface management for streaming multimedia. In *Wireless Communications and Networking Conference, 2002. WCNC2002. 2002 IEEE*, volume 2, pages 926 – 930 vol.2, March 2002.

[22] Davide Bertozzi, Anand Raghunathan, Luca Benini, and Srivaths Ravi. Transport protocol optimization for energy efficient wireless embedded systems. In *Proceedings of the conference on Design, Automation and Test in Europe - Volume 1*, DATE '03, pages 10706–, Washington, DC, USA, 2003. IEEE Computer Society.

[23] L. Cai and Yung-Hsiang Lu. Energy management using buffer memory for streaming data. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 24(2):141 – 152, Feb. 2005.

[24] Le Cai and Yung-Hsiang Lu. Dynamic power management using data buffers. In *Proceedings of the conference on Design, automation and test in Europe - Volume 1*, DATE '04, pages 10526–, Washington, DC, USA, 2004. IEEE Computer Society.

[25] S. Chandra. Wireless network interface energy consumption: implications for popular streaming formats. *Multimedia Systems*, 9(2):185–201, 2003.

[26] S. Chandra and A. Vahdat. Application-specific network management for energy-aware streaming of popular multimedia formats. In *the General Track, 2002 USENIX Annual Technical Conference*, pages 329 –342, Jun. 2002.

[27] Kyoung Youn Cho, S. Mitra, and E.J. McCluskey. Gate exhaustive testing. In *Test Conference, 2005. Proceedings. ITC 2005. IEEE International*, pages 7 pp. –777, Nov. 2005.

[28] Inc. Cisco Systems. Cisco visual networking index: Global mobile data traffic forecast update, 20122017, 2013. http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-520862.pdf.

[29] D.M. Cohen, S.R. Dalal, M.L. Fredman, and G.C. Patton. The AETG system: an approach to testing based on combinatorial design. *Software Engineering, IEEE Transactions on*, 23(7):437 –444, Jul 1997.

[30] M.B. Cohen, P.B. Gibbons, W.B. Mugridge, C.J. Colbourn, and J.S. Collofello. A variable strength interaction testing of components. In *Computer Software and Applications Conference, 2003. COMPSAC 2003. Proceedings. 27th Annual International*, pages 413 – 418, Nov. 2003.

[31] Radu Cornea, Alex Nicolau, and Nikil Dutt. Annotation based multimedia streaming over wireless networks. In *Embedded Systems for Real Time Multimedia, Proceedings of the 2006 IEEE/ACM/IFIP Workshop on*, pages 47 –52, Oct. 2006.

[32] Shuguang Cui, A.J. Goldsmith, and A. Bahai. Energy-constrained modulation optimization. *Wireless Communications, IEEE Transactions on*, 4(5):2349 – 2360, Sept. 2005.

[33] Y. Cui, X. Ma, H. Wang, I. Stojmenovic, and J. Liu. A survey of energy efficient wireless transmission and modeling in mobile cloud computing. *Mobile Networks and Applications*, pages 1–8, 2012.

[34] Yong Cui, Xiao Ma, Hongyi Wang, Ivan Stojmenovic, and Jiangchuan Liu. A survey of energy efficient wireless transmission and modeling in mobile cloud computing. *Mobile Networks and Applications*, 18:148–155, 2013.

[35] J. Czerwonka. Pairwise testing in the real world: Practical extensions to test-case scenarios. *Microsoft Corporation, Software Testing Technical Articles*, 2008.

[36] M. De Sanctis, E. Cianca, and V. Joshi. Energy efficient wireless networks towards green communications. *Wireless Personal Communications*, 59(3):537–552, 2011.

[37] F.R. Dogar, P. Steenkiste, and K. Papagiannaki. Catnap: exploiting high bandwidth wireless interfaces to save energy for mobile devices. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, pages 107–122. ACM, 2010.

[38] D. Feng, C. Jiang, G. Lim, L. Cimini, Jr., G. Feng, and G. Li. A survey of energy-efficient wireless communications. *Communications Surveys Tutorials, IEEE*, PP(99):1 –12, 2012.

[39] Monia Ghobadi, Yuchung Cheng, Ankur Jain, and Matt Mathis. Trickle: rate limiting youtube video streaming. In *Proceedings of the 2012 USENIX conference on Annual Technical Conference*, USENIX ATC'12, pages 17–17, Berkeley, CA, USA, 2012. USENIX Association.

[40] M. Gundlach, S. Doster, H. Yan, D.K. Lowenthal, S.A. Watterson, and Surendar Chandra. Dynamic, power-aware scheduling for mobile clients using a transparent proxy. In *Parallel Processing, 2004. ICPP 2004. International Conference on*, pages 557 – 565 vol.1, aug. 2004.

[41] Lei Guo, Enhua Tan, Songqing Chen, Zhen Xiao, Oliver Spatscheck, and Xiaodong Zhang. Delving into internet streaming media delivery: a quality and resource utilization perspective. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, IMC '06, pages 217–230, New York, NY, USA, 2006. ACM.

[42] Yong He and Ruixi Yuan. A novel scheduled power saving mechanism for 802.11 wireless lans. *Mobile Computing, IEEE Transactions on*, 8(10):1368 –1383, oct. 2009.

[43] M. Hoque, M. Siekkinen, and J. Nurminen. Energy efficient multimedia streaming to mobile devices - a survey. *Communications Surveys Tutorials, IEEE*, PP(99):1 –19, 2012.

[44] M.A. Hoque, M. Siekkinen, and J.K. Nurminen. On the energy efficiency of proxy-based traffic shaping for mobile audio streaming. In *Consumer Communications and Networking Conference (CCNC), 2011 IEEE*, pages 891 –895, Jan. 2011.

[45] M.A. Hoque, M. Siekkinen, and J.K. Nurminen. Tcp receive buffer aware wireless multimedia streaming - an energy efficient approach. In *Proceedings of the 23rd ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV '13, February 2013.

[46] Intelligent Broadband Networks. Global internet phenomena report, 2012. http://www.sandvine.com/downloads/documents/Phenomena_2H_2012/Sandvine_Global_Internet_Phenomena_Report_2H_2012.pdf.

[47] International Telecommunication Unio. Mobile traffic forecasts 2010-2020, 2011. http://groups.itu.int/LinkClick.aspx?fileticket=jUF0k4SHa-U%3D&tabid=1497&mid=5129.

[48] International Telecommunication Unio. Key global telecom indicators for the world telecommunication service sector, 2012. http://www.mincom.tn/fileadmin/PDF/Indicateurs_TIC/Rapports/ITU-D_ict_statistics.pdf.

[49] International Telecommunication Unio. The world in 2013: ICT facts and figures, 2013. http://www.itu.int/ITU-D/ict/facts/material/ICTFactsFigures2013.pdf.

[50] Christine E. Jones, Krishna M. Sivalingam, Prathima Agrawal, and Jyh Cheng Chen. A survey of energy efficient network protocols for wireless networks. *Wirel. Netw.*, 7(4):343–358, September 2001.

[51] Hongseok Kim, Chan-Byoung Chae, G. de Veciana, and R.W. Heath. A cross-layer approach to energy efficiency for adaptive mimo systems exploiting spare capacity. *Wireless Communications, IEEE Transactions on*, 8(8):4264 –4275, August 2009.

[52] Jari Korhonen and Ye Wang. Power-efficient streaming for mobile terminals. In *Proceedings of the International Workshop on Network and Operating Systems Support for Digital audio and Video*, NOSSDAV '05, pages 39–44, New York, NY, USA, 2005. ACM.

[53] Ronny Krashinsky and Hari Balakrishnan. Minimizing energy for wireless web access with bounded slowdown. In *Proceedings of the 8th Annual International Conference on Mobile Computing and Networking*, MobiCom '02, pages 119–130, New York, NY, USA, 2002. ACM.

[54] R. Kuhn, Yu Lei, and R. Kacker. Practical combinatorial testing: Beyond pairwise. *IT Professional*, 10(3):19 –23, May-June 2008.

[55] Kang Li, Charles Krasic, Jonathan Walpole, Molly Shor, and Calton Pu. The minimal buffering requirements of congestion controlled interactive multimedia applications. *Interactive Distributed Multimedia Systems*, pages 181–192, 2001.

[56] Mingzhe Li, Mark Claypool, and Robert Kinicki. Playout buffer and rate optimization for streaming over IEEE 802.11 wireless networks. *ACM Trans. Multimedia Comput. Commun. Appl.*, 5(3):26:1–26:25, August 2009.

[57] Xin Li, Mian Dong, Zhan Ma, and Felix C. A. Fernandes. GreenTube: power optimization for mobile videostreaming via dynamic cache management. In *20th ACM International Conference on Multimedia*, pages 279–288, 2012.

[58] Jiayang Liu and Lin Zhong. Micro power management of active 802.11 interfaces. In *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services*, MobiSys '08, pages 146–159, New York, NY, USA, 2008. ACM.

[59] Wenyu Liu, Xiaohua Li, and Mo Chen. Energy efficiency of mimo transmissions in wireless sensor networks with diversity and multiplexing gains. In *Acoustics, Speech, and Signal Processing, 2005. Proceedings. (ICASSP '05). IEEE International Conference on*, volume 4, pages iv/897 – iv/900 Vol. 4, March 2005.

[60] Yao Liu, Lei Guo, Fei Li, and Songqing Chen. An empirical evaluation of battery power consumption for streaming data transmission to mobile devices. In *Proceedings of the 19th ACM international conference on Multimedia*, MM '11, pages 473–482, New York, NY, USA, 2011. ACM.

[61] D. Marinov, A. Andoni, D. Daniliuc, S. Khurshid, and M. Rinard. An evaluation of exhaustive testing for data structures. Technical report, Technical Report MIT-LCS-TR-921, MIT CSAIL, Cambridge, MA, 2003.

[62] Guowang Miao, Nageen Himayat, Ye (Geoffrey) Li, and Ananthram Swami. Cross-layer optimization for energy-efficient wireless communications: a survey. *Wireless Communications and Mobile Computing*, 9(4):529–542, 2009.

[63] S. Mohapatra, N. Dutt, A. Nicolau, and N. Venkatasubramanian. Dynamo: A cross-layer framework for end-to-end qos and energy optimization in mobile handheld devices. *Selected Areas in Communications, IEEE Journal on*, 25(4):722 –737, May 2007.

[64] Shivajit Mohapatra, R. Cornea, H. Oh, K. Lee, M. Kim, Nikil Dutt, Rajesh Gupta, A. Nicolau, Sandeep Shukla, and Nalini Venkatasubramanian. A cross-layer approach

for power-performance optimization in distributed mobile systems. In *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, page 8 pp., April 2005.

[65] Shivajit Mohapatra, Radu Cornea, Nikil Dutt, Alex Nicolau, and Nalini Venkatasubramanian. Integrated power management for video streaming to mobile handheld devices. In *Proceedings of the Eleventh ACM International Conference on Multimedia*, MULTIMEDIA '03, pages 582–591, New York, NY, USA, 2003. ACM.

[66] K. Naik. *A survey of software based energy saving methodologies for handheld wireless communication devices*. Technical Report 2010, Department of Electrical and Computer Engineering, University of Waterloo, 2010.

[67] Adel Noureddine, Romain Rouvoy, and Lionel Seinturier. A review of middleware approaches for energy management in distributed environments. *Software: Practice and Experience*, pages n/a–n/a, 2012.

[68] B. O'hara and A. Petrick. *IEEE 802.11 handbook: a designer's companion*. Standards Information Network, 2005.

[69] Rajesh Palit, Renuka Arya, Kshirasagar Naik, and Ajit Singh. Selection and execution of user level test cases for energy cost evaluation of smartphones. In *Proceedings of the 6th International Workshop on Automation of Software Test*, AST '11, pages 84–90, New York, NY, USA, 2011. ACM.

[70] K. Pentikousis. In search of energy-efficient mobile networking. *Communications Magazine, IEEE*, 48(1):95 –103, January 2010.

[71] X. Perez-Costa and D. Camps-Mur. IEEE 802.11E QoS and power saving features overview and analysis of combined performance [accepted from open call]. *Wireless Communications, IEEE*, 17(4):88 –96, August 2010.

[72] N. Pettis, L. Cai, and Yung-Hsiang Lu. Statistically optimal dynamic power management for streaming data. *Computers, IEEE Transactions on*, 55(7):800 – 814, July 2006.

[73] N. Pettis, Le Cai, and Yung-Hsiang Lu. Dynamic power management for streaming data. In *Low Power Electronics and Design, 2004. ISLPED '04. Proceedings of the 2004 International Symposium on*, pages 62 –65, Aug. 2004.

[74] M Rosu, C Olsen, Lu Luo, and Chandrasekhar Narayanaswami. The power-aware streaming proxy architecture. In *Proceedings of BROADNETS*, 2004.

[75] Curt Schurgers, Vijay Raghunathan, and Mani B. Srivastava. Modulation scaling for real-time energy aware packet scheduling. In *Global Telecommunications Conference, 2001. GLOBECOM '01. IEEE*, volume 6, pages 3653 –3657 vol.6, 2001.

[76] Curt Schurgers, Vijay Raghunathan, and Mani B. Srivastava. Power management for energy-aware communication systems. *ACM Trans. Embed. Comput. Syst.*, 2(3):431–447, August 2003.

[77] Prashant J. Shenoy and Peter Radkov. Proxy-assisted power-friendly streaming to mobile devices. pages 177–191, 2003.

[78] M. Siekkinen, M. Ashraful, J.K.N. Hoque, and M. Aalto. 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*, MoVid '13, February 2013.

[79] Enhua Tan, Lei Guo, Songqing Chen, and Xiaodong Zhang. Psm-throttling: Minimizing energy consumption for bulk data communications in wlans. In *Network Protocols, 2007. ICNP 2007. IEEE International Conference on*, pages 123 –132, oct. 2007.

[80] Shiao-Li Tsao and Chung-Huei Huang. A survey of energy efficient MAC protocols for IEEE 802.11 WLAN. *Computer Communications*, 34(1):54 – 67, 2011.

[81] Yong Wei, S.M. Bhandarkar, and S. Chandra. A client-side statistical prediction scheme for energy aware multimedia data streaming. *Multimedia, IEEE Transactions on*, 8(4):866 –874, Aug. 2006.

[82] H. Yan, R. Krishnan, S.A. Watterson, D.K. Lowenthal, K. Li, and L.L. Peterson. Client-centered energy and delay analysis for TCP downloads. In *Quality of Service, 2004. IWQOS 2004. Twelfth IEEE International Workshop on*, pages 255 – 264, June 2004.

[83] H. Yan, S.A. Watterson, D.K. Lowenthal, K. Li, R. Krishnan, and L.L. Peterson. Client-centered, energy-efficient wireless communication on IEEE 802.11b networks. *Mobile Computing, IEEE Transactions on*, 5(11):1575 –1590, Nov. 2006.

[84] Jinyao Yan, W. Muhlbauer, and B. Plattner. Analytical framework for improving the quality of streaming over tcp. *Multimedia, IEEE Transactions on*, 14(6):1579 –1590, Dec. 2012.

[85] Fan Zhang and SamuelT. Chanson. Proxy-assisted scheduling for energy-efficient multimedia streaming over wireless lan. In Raouf Boutaba, Kevin Almeroth, Ramon Puigjaner, Sherman Shen, and JamesP. Black, editors, *NETWORKING 2005. Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications Systems*, volume 3462 of *Lecture Notes in Computer Science*, pages 980–991. Springer Berlin Heidelberg, 2005.