Application-Centric Wi-Fi Energy Management on Smart Phone

Jian Li
Computer Science and Engineering
POSTECH, Pohang, South Korea
Email: gunine@postech.ac.kr

Jin Xiao and James Won-Ki Hong
Division of IT Convergence Engineering
POSTECH, Pohang, South Korea
Email: {jinxiao, jwkhong}@postech.ac.kr

Raouf Boutaba
School of Computer Science
University of Waterloo, Canada
Email: rboutaba@cs.uwaterloo.ca

***Abstract*—Vast majority of the services running on the smart phone today are networked in that significant amount of communication is required. Smart phone energy expenditure due to Wi-Fi communications constitutes significant portion of the battery discharge. In this paper, we first investigate the key factors influencing Wi-Fi energy consumption, and propose three energy management schemes: 1) Dynamic control of Wi-Fi on/off interface; 2) improve communication efficiency via application packing; and 3) elongation of Wi-Fi Power Save Mode (PSM) via application alignment under mixed application workload. We also design and test our solution as a device-side application utilizing general system process scheduling and network firewall techniques. As the result, our solution is easy to deploy, applicable to most mobile devices, and we explicitly tackle the challenging case of download management. Through extensive experimentation and solution prototyping, we show the effectiveness of device side Wi-Fi energy management and the importance of considering application characteristics.**

***Index Terms*—smart phone, energy management, design and experimentation**

I. Introduction

Recently, the mobile device have proliferated as a common communication and data service platform. Vast majority of the services running on the smart phone today are networked in that significant amount of communication is required. Since 3G (i.e., HSPA+ and EVDO Rev B) and 4G (i.e., WiMAX and LTE) telecommunication networks charges data communication at a premium rate, majority of the smart phone users leverage the Wi-Fi data connectivity whenever possible. Furthermore, most of the smart phone applications are relying on continuous internet connectivity (i.e., social network service and location based service). Consequently, smart phone energy expenditure due to Wi-Fi communications constitutes significant portion of the battery discharge, especially for the download workload (see Figure 1). Therefore, it is important to leverage energy saving hardware solutions coupled with smart application level control as to facilitate effective Wi-Fi energy management for smart phones. At the hardware level, Power Save Mode (PSM) is explicitly designed to achieve energy saving efficiency [1]. Vast majority of the energy saving solutions in literature attempts to leverage the PSM feature, but do so mostly at the network level, without good understanding of the application requirements and characteristics. As a result, the proposed schemes tend to be either too soft (i.e., do not provide sufficient energy saving under a realistic workload) or too hard (i.e., breaks user experience due to excessive delays). Furthermore, to the best of our knowledge, there is no effective device side solutions for Wi-Fi energy management.

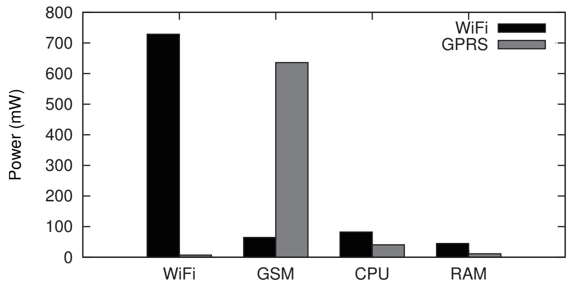


Fig. 1. Power consumption by smartphone for file downloading [2]

In this paper, we first investigate the key attributes that affects Wi-Fi energy consumption during communications including network RSSI, application throughput and application workload characteristics. Based on our investigation, we propose three device-side energy saving schemes that address a number of key issues in Wi-Fi energy management today: dynamic control of Wi-Fi on/off interface, extension of Wi-Fi PSM mode for energy shaving, and scheduling of Wi-Fi communication based on user application specifics and user experience constraints. We also design and test device-side operational control mechanisms that can realize our management schemes, in particular we solve the difficult case of down stream management at device side. By implementing our solutions as an Android application, we show that we can achieve significant Wi-Fi energy saving under different mix of user applications and in non-trivial use cases.

The remainder of this paper is organized as follows. Section II presents related works on energy measurement and management in mobile device, and Section III presents a detail experiment study on the key factors to consider for Wi-Fi energy management. In Section IV, we present our energy management schemes and in Section V, we present two effective application management mechanisms. Section VI reports on our experiment results based on a prototype implementation. Section VII concludes the paper.

II. Related work

In [3], energy usage pattern are analyzed according to hardware components in mobile device. Based on result, the authors proposed an energy usage estimation model and a methodology for optimize the energy consumption. The hardware components they focused on are CPU and LCD. On-Demand CPU Governor, which is originally included as a CPU scheduler in Linux system, is used to dynamically change the CPU frequency according to current CPU utilization rate. The authors extended the On-Demand CPU Governor to not only considers the CPU utilization rate when it performs scaling, but also to monitor the status of LCD display. They assume that background applications remain active, even when the phone goes into the hibernate mode (display off). Since maintaining the User eXperience (UX) is not important in hibernation, energy saving can be obtained by lowering the CPU frequency.

In [4], the authors proposed an energy measurement model for monitoring the energy usage of various hardware components, including CPU, LCD, GPS, Audio, Wi-Fi and 3G in real time. They provided two views of energy consumption: 1) per application view and 2) overall energy consumption. The proposed energy usage model is evaluated by exploiting an external energy measurement device. The authors also implemented an Android application called PowerTutor [5] based on the proposed measurement model that is shown to provide good energy measurement of Wi-Fi component. We leverages PowerTutor for our energy studies by extending their measurements to include down streaming.

In [6], the authors proposed Catnap, which exploits the bandwidth discrepancy between wireless Access Point (AP) (low) and mobile phone (high) to perform the energy reduction in down streaming case. The slow speed between AP and server may enforce the client to be always in active mode, so that large amount of energy is wasted. The authors introduced a proxy-based modification to access point which serves as an external data buffer to leverage the slow link speed between AP and server. The main drawback of this approach is that it requires proxy to be deployed at access point, stores and may expose user data at shared public locations, and assumes that wireless channel has higher speed than wired part which is not always true around the world (e.g., South Korea).

III. Analysis of Wi-Fi Energy Consumption

In this section, we will present a number of key experimental observations concerning Wi-Fi energy consumption in Android: the relation between RSSI and energy, the effect of application throughput on energy, and the application characteristics that affects energy management. Based on these observations, we present three energy management schemes in Section IV. For energy measurement, we use PowerTutor deployed on HTC Rider [7] smart phone running Android Gingerbread 2.3.7. HTC Rider is designed with the BCM4329 [8] Wi-Fi communication module. From our experiment results, we find that energy consumption patterns for data uploading and downloading are generally identical. Since data downloading is by far the majority of Wi-Fi data access on smart phones today, we present our observations based on experiments conducted on data downloading in this section.

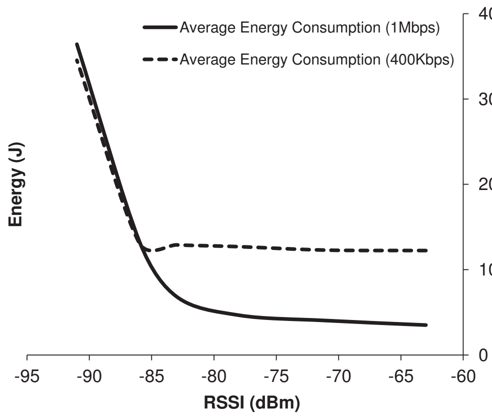


Fig. 2. Correlation between RSSI and Energy in Two Data Rates

A. RSSI and energy consumption

Received Signal Strength Indicator (RSSI) is a metric that quantifies the power level present in a received radio signal. The measurement unit is dBm. The RSSI value of the associated Wi-Fi interface is readily obtainable from Android kernel. In general, the RSSI value (in dBM) is inversely related to Euclidean distance, meaning that RSSI decreases linearly with respect to increasing distance from the transmitter. RSSI is also inversely related to environmental noise, but the exact relation is uncertain. In summary, a higher RSSI value denotes better signal reception strength and therefore higher channel capacity. *Channel capacity* is a indicator if the rate at which digital binary bits are transmitted or received across the underlying physical communication channel (the air medium in this case). When RSSI is low, data loss and error rate increases. To compensate, the transmitter must lower the data rate and therefore the overall channel capacity is reduced.

Figure 2 shows total energy consumption of downloading a fixed amount of data (i.e., 60MB) under varying RSSI values. Two sets of experiments are conducted. In the first set, the file is downloaded from a server with serving capacity of 1Mbps. In the second set, the file is downloaded from a server with serving capacity of 400kbps. We observe a clear threshold based relationship between RSSI and energy consumption. When the RSSI is above threshold (e.g., around -85dBm), the amount of energy expenditure is similar across different RSSI values. This indicates that the channel capacity above RSSI threshold is at maximum and is consistent. On the other hand, when the RSSI value is below threshold, we see exponential increase in energy expenditure due to the exponential decrease in channel capacity. Furthermore, we observe that this threshold value is independent of application throughput. Therefore, if we can ascertain the threshold RSSI value accurately in practice, we can achieve significant energy savings. However, this threshold value is subject to the particularities of the underlying communication channel conditions and the environment, and therefore there is not a single fixed threshold value that can be set. Accordingly, we define a mechanism to facilitate Wi-Fi control in Section IV-A, and provide an energy efficient Wi-Fi control mechanism.

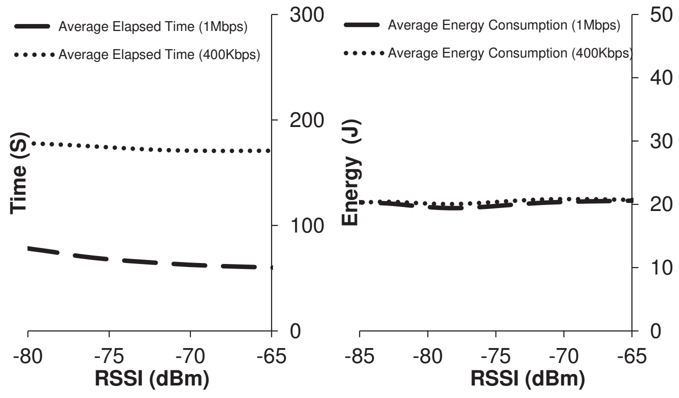


Fig. 3. Correlation between RSSI and Transmission Time in Two Data Rates

B. Application throughput and energy consumption

Wi-Fi communication module provides some means of dealing with varied application throughput. In the most simple case, when there are no applications requiring data communication, the Wi-Fi module can be put into off mode whereby the energy consumption is practically zero. However, this scheme is not very efficient in practice as when application requires communication, which may occur at any given time, there is a significant overhead for the Wi-Fi module to turn itself back on and to scan and associate with an access point. This lag time is in the range of 4 seconds or more, which is considered intolerable with respect to satisfactory user experience.

Fortunately, today's Wi-Fi radio module of a smart phone also provides multiple communication mode during the on state explicitly designed with energy conservation in mind. For instance, the widely used BCM4329 communication module in the on state has a Constantly Awake Mode (CAM) which consumes 1120mW or more and a Power Save Mode (PSM) which consumes 20mW. The caveat of PSM is that the application throughput must be extremely low (approx. 8 or less packets per second [4]) for the device to remain in PSM. In literature, one of the common strategies to employ for Wi-Fi energy saving is to keep the device in PSM mode for as long as possible. However, it is still an open question concerning the relation between application throughput and energy consumption rate in CAM mode. Or in another word, is there an energy efficient optimal throughput rate in CAM mode? and if so, is this rate RSSI dependent?

To answer these questions, we've conducted a series of experiments as shown in Figure 3. We conducted two sets of experiments downloading from two data servers with varied serving capacity as before. And we've isolated our experiment in the regions of RSSI above threshold values such that we can ensure constant channel capacity. The left of Figure 3 shows the total transmission time it takes to download a fixed data from the servers. It is clear and intuitive to see that downloading the same file at higher serving capacity requires longer time and hence more energy expenditure. This is independent of the RSSI values at above threshold. The right of Figure 3 shows the energy consumption rate (J/s) of the two experiment sets. Interestingly, the energy consumption rates in CAM do not differ significantly under extremely varied application throughput. And again, this is independent of RSSI values above threshold. It is then logical to see that the total energy consumption of data communication is strictly dependent on the total time of transmission. Therefore, to achieve good energy saving, we should always aim to perform data transmission at maximum throughput whenever possible. According to this observation, we have designed data packing scheme in Section IV-B that not only leverage the energy saving of PSM but also the maximum bursting characteristics of CAM.

C. Understanding the applications

To date, vast majority of the energy saving solutions in literature manage the Wi-Fi traffic purely at the network level, without understanding the application requirements and characteristics. As a result, the proposed schemes tend to be either too soft (i.e., do not provide sufficient energy saving under a realistic workload) or too hard (i.e., breaks user experience due to excessive delays).

Therefore it is important to understand what are the critical application characteristics and how they impact and constrain the design of energy saving mechanisms. Figure 4 showcases the workloads of different application types: AndFTP [9] is a data transfer application; WhatsApp [10] is an instant messaging application; Facebook [11] is a web based social networking service application; PPSTV [12] is a P2P-based video streaming application; DropBox [13] is a file synchronization service application; Remote Desktop [14] is a graphical desktop sharing service application. After analyzing most of the common application types used on smart phone today, we summarize below a number of important characteristics that affects energy saving designs.

Delay sensitivity: some applications are very delay sensitive while others are more delay tolerant. For instance, Facebook, PPSTV and Remote Desktop are delay sensitive in that the user interactivity and experience is strongly affected by delay. Other applications may have different degree of tolerance to delay. For instance WhatsApp can tolerate a few seconds of delay while AndFTP and Dropbox are generally very delay tolerant. Delay tolerant applications can be delayed for finite duration in order to better coordinate data transmissions for energy saving. However, when there is a mix of delay sensitive applications and delay tolerant applications, the transmission behavior of delay sensitive applications can be used to "cue in" packing schedules of the delay tolerant applications. We call this process alignment and is presented in Section IV-C.

Expected throughput: understanding the expected peak throughput of applications helps to better schedule application

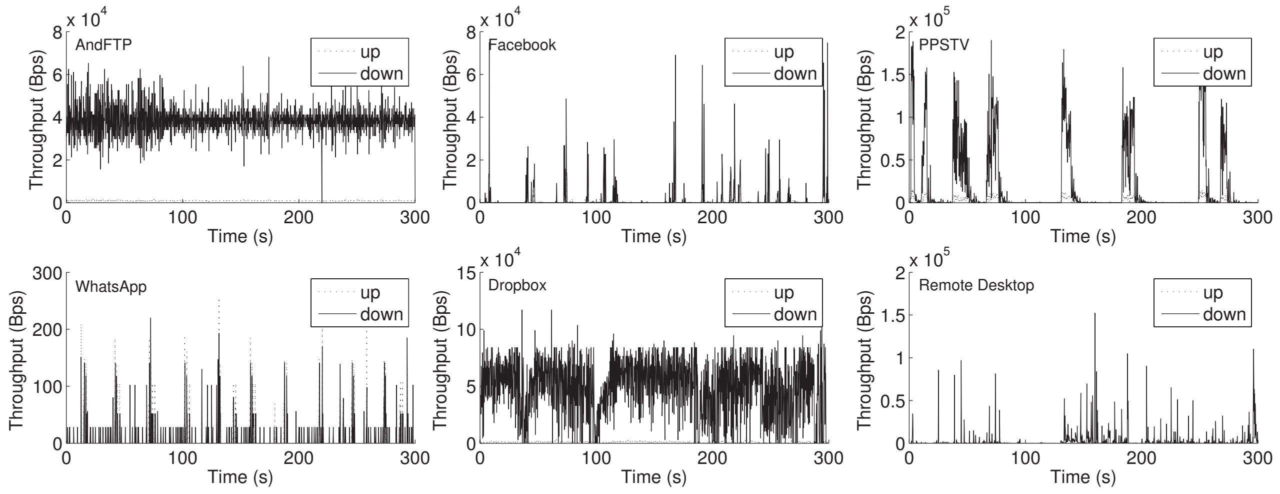


Fig. 4. Workload of Example Application Types

transmissions. Generally, different types of applications have different level of expected peak throughput. For instance, WhatsApp has very low expected peak throughput, while PPSTV is much higher. Yet other applications such as Facebook has more varied peak throughput depending on the actual page content.

Burst cycle: data transfers are bursty. We observe that majority of the applications have distinct burst cycles due to the existence of “data units”. A data unit is an atomic set of data and/or transactions that must be completed for the application to deliver its function. For instance, WhatsApp’s data unit is the delivery of a message; Facebook’s data unit is the loading of a page; PPSTV’s transmission is related to video frame grouping and encoding, etc. Knowing the burst cycle of an application helps to better determine the expected throughput of the application and the required monitoring period for estimating true throughput (Section IV-A).

Session preservation: TCP flows are largely session based, which is by far the most used transport protocol for smart phone applications [15]. Even for UDP based transmissions such as video streaming and online gaming, there is often the concept of a session that is re-enforced by an application level protocol such as RTP. It is then important to understand, how long can a transmission be delayed without breaking a session, as failure to do so result in service unavailability and/or wasted work to the user. The exact requirement of session preservation also affects the choice of management mechanisms as we will discuss in Section V.

IV. Energy Saving Designs for Wi-Fi Transmissions

In this section, we present three energy saving designs for Wi-Fi transmissions based on the analysis results we have discussed in Section III.

A. Dynamic Wi-Fi on-off control

The goal of Wi-Fi on-off control is to turn off Wi-Fi communication when the RSSI value is below threshold. However, as we have discussed in Section III-A, the threshold RSSI value is environment dependent. Furthermore, even at low RSSI values, applications that has low expected throughput (e.g., WhatsApp) are not adversely affected. Therefore, we establish the control parameter for when to turn off Wi-Fi as $\alpha = \frac{Th}{E_{Th}}$ where Th is the true throughput of the application at current RSSI value and E_{Th} is the expected throughput of the application. For delay sensitive applications, α should be large (e.g., 0.8), while for very delay tolerant applications such as AndFTP α may be very small.

It follows then that we require a measure of true throughput of the wireless channel. Since RSSI value is related to transmission rate, can we determine true throughput from RSSI? The top left of Figure 5 shows the relation as measured in our experiments based on a simple one phone to one access point topology. We observe that although there is a strong correlation between RSSI and link speed in such simple scenario, the relation is not nearly precise enough for decision making. For instance, link speed fluctuates significantly within RSSI value range -70dBm and -85dBm. To make matter worse, link speed is a measure of the physical channel raw data rate, not the actual application level throughput which is dependent on both the protocol overhead, error re-transmission rate, as well as the medium contention rate. None of the above values are particularly easy to obtain from the device side without significant monitoring overhead. These discoveries forced us to examine an alternative solution: to deduce the application’s true peak throughput against its expected peak throughput by passive listening. In essence, when the application is communicating via Wi-Fi, we sample the observed peak throughput within a sampling window. To determine the correct sampling window size, we run a series of experiments under 1 second, 3 seconds, 5 seconds, 10 seconds and 20 seconds window sizes as shown in Figure 5. Intuitively, we see that the accuracy of the true throughput determination increases with larger sampling window size. In investigating the correct window size setting, we find that it is directly tied to the burst cycle length

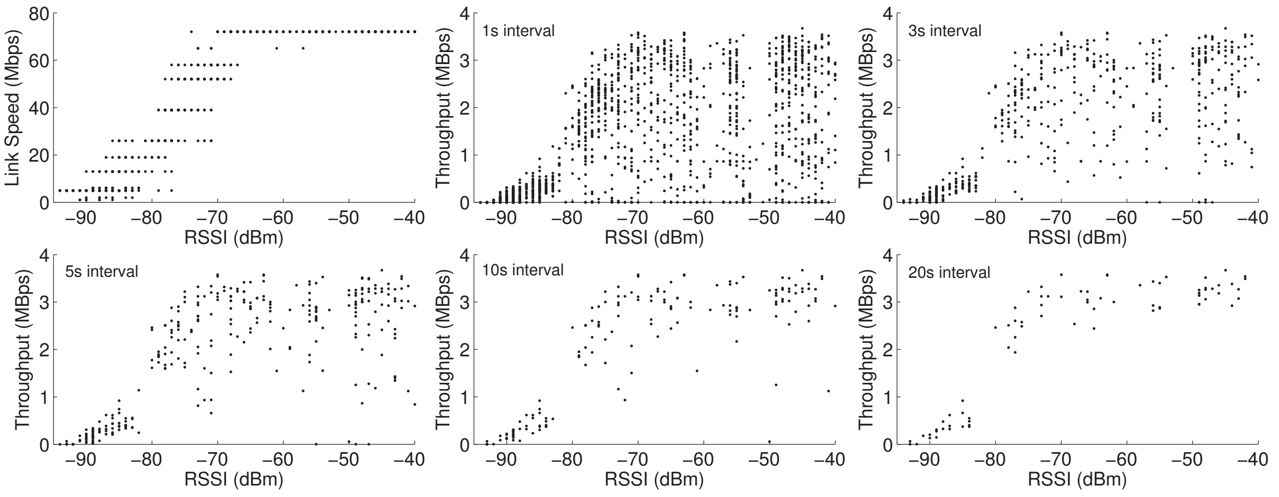


Fig. 5. Relation between RSSI, Linkspeed and Throughput

of the application. In the particular instance shown Figure 5, AndFTP is the example application, which has an average burst cycle length of around 10 seconds. Hence it appears that approximately two times the burst cycle length is necessary to obtain a good estimate of the true application throughput. Accordingly, we are then able to cheaply compute α to be used as the Wi-Fi on-off control parameter. We further note that when a decision is made for Wi-Fi to be turned off. We do not actually turn the Wi-Fi off at the hardware level, but instead cuts off the application's ability to communicate. The reason for not turning Wi-Fi physically off is two fold: 1) we want to still be able to monitor the Wi-Fi channel such that the transmission may resume when channel condition improves; and 2) the cost of channel scan and re-association is very energy consuming (nearly 5 times the energy consumption of CAM [16]). We present the mechanisms for application control in Section V.

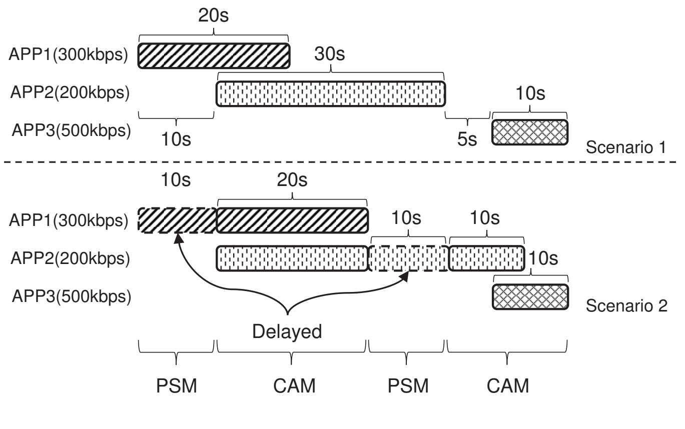


Fig. 6. Example of Packing

B. Packing for delay tolerant applications

For delay tolerant applications, we can deliberately delay its transmission for certain amount of time (determined by an application's session preservation) in hope to allow multiple applications to transmit at the same time. We term this "packing" mechanism. The goal of packing is to extend the Wi-Fi module's PSM time and to achieve maximum bursting when transmitting.

The procedure for packing is as follows: when a delay-tolerant application requires communication, its transmission is delayed for fixed amount of time if its expected throughput is below the true channel capacity. This delay is terminated whenever the total expected throughput of the applications exceeds the true channel capacity (i.e., the pack is full) or when an application cannot be delayed any further (as to preserve session integrity).

In the following example, three delay tolerant applications intend to transmit at varied times each with a delay deadline of 10 seconds and true throughput of 500kbps (see Figure 6). Top of Figure 6 shows the transmission sequence without packing. The Wi-Fi module spends 5 seconds in PSM and the rest of the time in CAM. The bottom of Figure 6 shows the result of packing. When APP1 intends to communicate, it is delayed until the arrival of APP2's request for communication. Both use the channel at this point since APP1 cannot be delayed any further. After APP1 finishes, APP2 is delayed for 10 seconds after which it again starts to communicate. APP3 arrives and is not delayed since the overall expected throughput of APP1 and APP2 has already exceeded the pre-defined true throughput. Overall, the Wi-Fi module spent 20 seconds in PSM.

C. Alignment for mixed application types

When there is a mixture of delay sensitive and delay tolerant applications, we do not delay the communication of delay sensitive applications. But rather, we try to affix the communication of delay tolerant applications as to "align" them into the communication patterns of the delay sensitive applications.

Figure 7 shows an example. APP1 is delay sensitive and APP2 is delay tolerant. Instead of having them each communi-

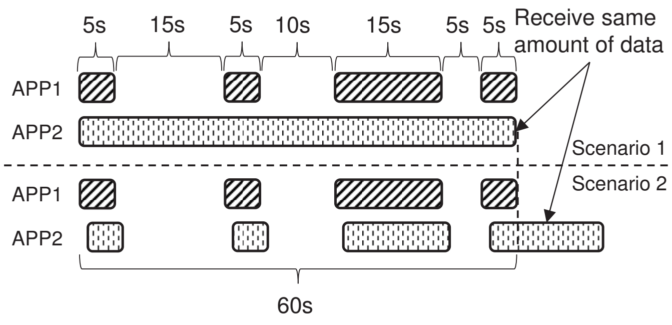
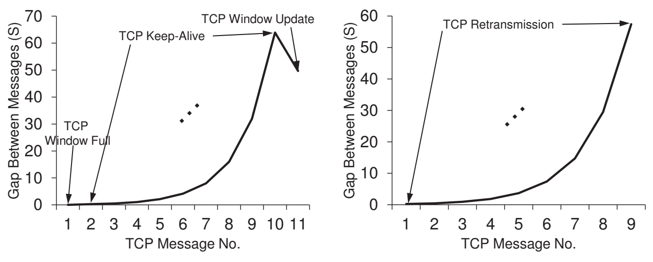


Fig. 7. Example of Alignment



(a) Process manipulation scheme (b) Traffic shaping scheme

Fig. 8. Issued TCP messages in two controlling schemes

cate independently (top of Figure 7), we delay the transmission of APP2 intermittently such that APP2 is in-sync with the communication of APP1. Due to the delay introduced, APP2 will take longer to finish than the un-managed case.

V. Enabling Application Communication Management in Android

In this section, we discuss how application communications can be delayed and scheduled in practice. We focus our discussion on downloading by the smart phones because 1) downloading is by far the pre-dominate communication of smart phones; 2) mechanism for enabling download management at device side is much harder to achieve than for upload management from device. The same technique used for managing download is readily applicable for managing upload. We have developed two mechanisms for application management: manipulation of the process life-cycle (kernel space) and traffic shaping via firewall (network space):

- **Manipulation of process life-cycle**: processes can be suspended and resumed in kernel space. This mechanism is designed for efficient CPU resource sharing and scheduling. We exploit this mechanism for our application control purpose. It works because once the process is suspended, it will no longer receive or transfer the data. And the TCP flow control mechanism ensures that no further data traffic will be sent from the server.
- **Traffic shaping via firewall**: exploiting the firewall policy to shape the traffic is one of the well know technique for controlling traffic.

We experimented the process manipulations scheme on PPSTV. Process suspension is performed during video playing back. Once PPSTV was suspended, there was no incoming traffic to the phone, but a few outgoing traffic from the phone is made for sustaining the TCP session. After 3 minutes of delay, we resume PPSTV at the process level and find that the down stream service resumed successfully with little delay. Such recovery is possible, because PPSTV uses video buffer to store arriving packets. On resume, it is able to continue its data downloading immediately since the TCP session is preserved. We had similar success with other types of delay tolerant applications such as DropBox, AndFTP, etc.

This behavior highlights some interesting properties regarding TCP which is used by most of the smart phone applications today. TCP traffic control involves two types of control scheme, 1) Congestion control and 2) Flow control.

In the process manipulation scheme, the TCP flow control scheme is exploited for controlling traffic. Once the process resides in the receiver is suspended, it cannot consume any newly arriving packets, so that the receiver buffer becomes full immediately. Consequently, the receiver starts to issue *TCP Window Full* message along with *TCP Zero Window* to inform the sender to stop sending data. In Figure 8, left figure shows the overall TCP messages yielded while the Android application process is suspended. The *TCP Window Full* message is issued one second after the process is suspended, and this message indicates there is no available buffer space for storing incoming packet in receiver perspective. The *TCP Keep-Alive* message follows the *TCP Window Full* message, and this message is for informing the sender that the receiver was not terminated and was keeping processing the received data. Note that *TCP Zero Window* message is also issued to the sender along with *TCP Keep-Alive* message, to inform the sender to stop sending data. However, such event message definitely yields another overhead to the network. In order to mitigate such overhead, TCP uses the build-in exponential back-off time scheme. According to this scheme the event message such as *TCP Keep-Alive* is issued in exponentially increased time frame. Once the process is resumed, the receiver automatically detects the state, and issues *TCP Window Update* message spontaneously to the sender to inform that the receiver is ready to receive data.

In the traffic shaping scheme, the TCP congestion control scheme is exploited for traffic controlling purpose. Once traffic shaper starts to constrain all the incoming and outgoing traffic for one application via policy rule, all the ACK messages generated by receiver are blocked out by traffic shaper. The sender assumes that the receiver is failed to receive the data for some reasons, so that it starts to perform the data re-transmission. In Figure 8, right figure shows the overall TCP messages yielded when the traffic shaping scheme is enabled. Similar to the *TCP Keep-Alive* message in process manipulation scheme, the *TCP Retransmission* message is also issued in exponentially increased time frame. Note that in this scheme, even if the traffic shaping is disabled at some points, the receiver can only receive the data until the arrival of the next *TCP Retransmission* message.

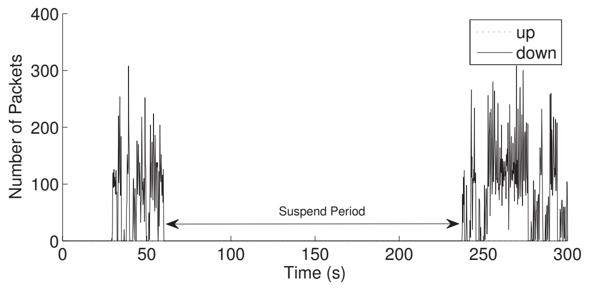


Fig. 9. Traffic pattern of process manipulation scheme on AndFTP

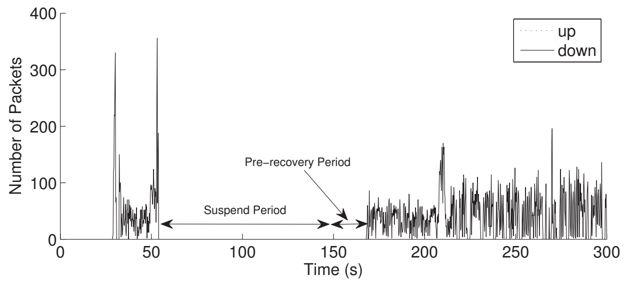


Fig. 10. Traffic pattern of traffic shaping scheme on AndFTP

To validate our analysis result and compare the two schemes, we conducted experiments using AndFTP. The first experiment was performed by using process manipulation scheme (see Figure 9), and the second experiment was performed by using traffic shaping scheme (see Figure 10). In the first experiment, we set the suspend period as 3 minutes, while for the second experiment, we suspended for 90 seconds. With process manipulation scheme, we did not observe any recovery lag, while in traffic shaping scheme, it incurred lag of around 17 seconds before resuming, and as we increased the length of suspend, the time required for recovering increased as well.

We conclude that process manipulation scheme is effective for delay tolerant applications which typically have background synchronization and buffering functionalities. But for user interactive applications (even if they are somewhat delay tolerant such as WhatsApp), traffic shaping is more appropriate even though it is less efficient, because process suspense will render the application unresponsive to the users and significantly disrupt user experience.

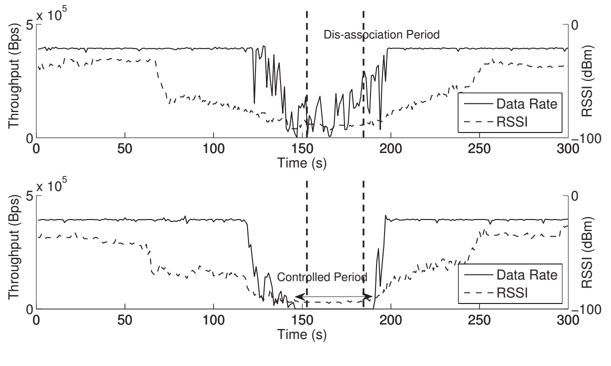
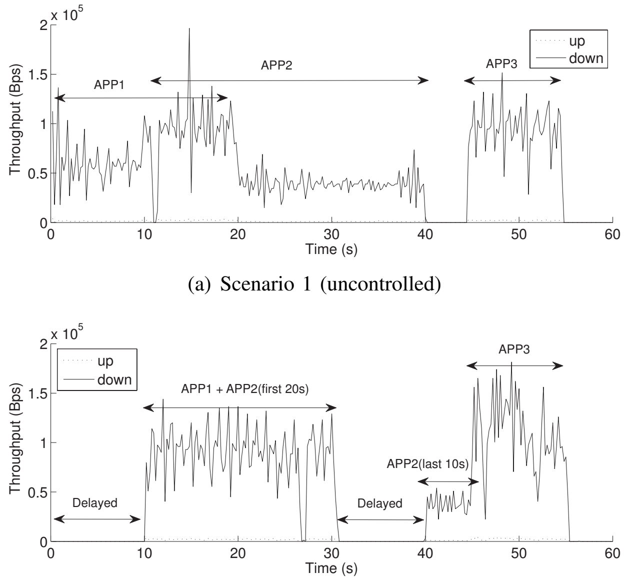


Fig. 11. Throughput comparison in different signal strength zone

VI. Experiment Studies

In this section, we evaluate the effectiveness of our energy saving mechanisms in practice. To do so, we have implemented the management schemes in an application called Automated Application-Aware Battery Manager (AAA BattMan). Current implementation of AAA BattMan is targeted for Android platform and it uses network monitor implemented as a kernel module. Since the current version of Android (Gingerbread 2.3.7) does not provide any information on packet transmission by each application, we customized the kernel source(2.6.35.14) from htcdev [17] to add the network monitoring capability.



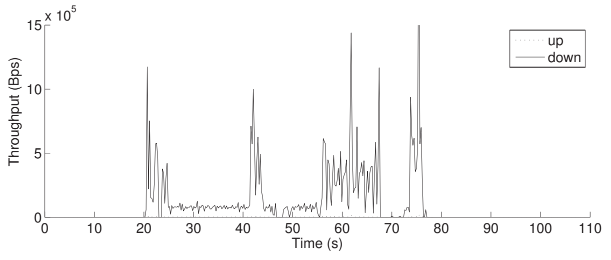
(a) Scenario 1 (uncontrolled)

(b) Scenario 2 (controlled)

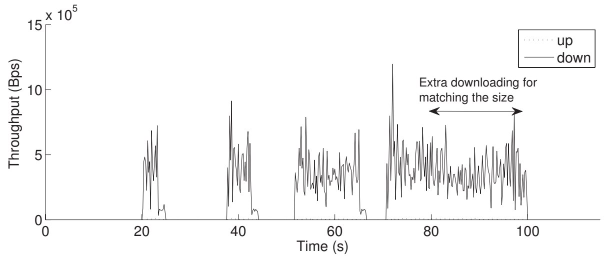
Fig. 12. Throughput comparison of the second experiment

Three experiments are performed, and each of which corresponds to the that of proposed energy management schemes. In each case, we compare the result of not running AAA BattMan (uncontrolled) with running AAA BattMan (controlled).

In the first experiment, we examine the effect of dynamic Wi-Fi on-off control. We let the smart phone download a fixed size file from server using AndFTP while moving to and from a good signal reception zone and a bad signal reception zone. We set the $\alpha = 0.016$ which is very generous (i.e. Wi-Fi cuts off when true throughput is 1.6% of the expected throughput). We see that when AAA BattMan is enabled, the Wi-Fi is indeed cut off (controlled period) when α becomes too low. In the uncontrolled case, two AP dissociations occurred in the bad reception zone (disassociation period), when the controlled case avoided these dissociations. In comparison, the uncontrolled case took 5 minutes and 8 seconds to obtain the file while our controlled case took 5 minutes and 28 seconds. The throughput comparison has been made for the two cases and the result is depicted in Figure 11. We also compare the total energy expenditures, and we achieve round 28% energy reduction.



(a) Scenario 1 (uncontrolled)



(b) Scenario 2 (controlled)

Fig. 13. Throughput comparison of the third experiment

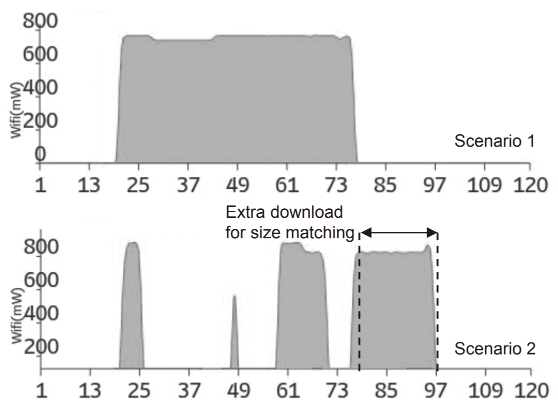


Fig. 14. The comparison on energy consumption for the third experiment

In the second experiment, we performed packing on delay tolerant applications using the example scenario shown in Figure 6. Three different applications are used each of which requires 300kbps, 200kbps and 500kbps throughput respectively, and delay deadline of 10 seconds. The overall throughput of this experiment is shown in Figure 12. In the controlled case, we have 20 seconds PSM duration and we achieved 23% energy reduction by using AAA BattMan.

In the third experiment, we have a mix of delay sensitive and delay tolerant applications as depicted in example 7. In the controlled case, APP2 is delayed intermittently in order to sync with APP1's transmission. The overall throughput by the APP1 and APP2 is shown in Figure 13, and for this example, we show the energy expenditure result as captured by PowerTutor in Figure 14. In the controlled case, we have 30s PSM duration (notice the drop in energy consumption in PSM), while we have zero PSM duration in the uncontrolled case. Overall, AAA BattMan saved us 22.6% energy this experiment.

VII. Conclusion

In this paper, we investigate the key factors influencing Wi-Fi energy consumption: RSSI, throughput and application characteristics. We propose effective energy management schemes for dynamic Wi-Fi on-off control, packing for delay tolerant applications, and alignment for mix of different application types. We also present device-side mechanisms for realizing our designs. Through extensive experimentation and solution prototyping, we show the effectiveness of device side Wi-Fi energy management and the importance of considering application characteristics. As future work, we will further mature our design and conduct more extensive experiments.

Acknowledgment

This research was supported by World Class University program funded by the Ministry of Education, Science and Technology through the National Research Foundation of Korea (R31-10100) and also supported by the MKE (The Ministry of Knowledge Economy), Korea, under the ITRC (Information Technology Research Center) support program supervised by the NIPA (National IT Industry Promotion Agency) (NIPA-2012-H0301-12-3002)

References

[1] E. Rozner, V. Navda, R. Ramjee, and S. Rayanchu, "Napman: Network-assisted power management for wifi devices," in *Proceedings of the 8th international conference on Mobile systems, applications, and services (MobiSys 2010)*, 2010, pp. 91–106.
[2] A. Carroll and G. Heiser, "An analysis of power consumption in a smartphone," in *Proceedings of the 2010 USENIX conference on USENIX annual technical conference (USENIXATC 2010)*, 2010.
[3] A. Shye, B. Scholbrock, and G. Memik, "Into the wild: Studying real user activity patterns to guide power optimizations for mobile architectures," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 42)*, 2009, pp. 168–178.
[4] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Proceedings of the 8th IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis (CODES/ISSS 2010)*, 2010, pp. 105–114.
[5] *PowerTutor*. [Online]. Available: http://www.powertutor.org/
[6] 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 (MobiSys 2010)*, 2010, pp. 107–122.
[7] *HTC Evo 4G+ (Rider) Hardware Specification*. [Online]. Available: http://www.htc.com/kr/smartphones/htc-evo-4g-plus/
[8] *Data sheet for Broadcom BCM4329 Communication Module*. [Online]. Available: http://ko.broadcom.com/products/Bluetooth/Bluetooth-RF-Silicon-and-Software-Solutions/BCM4329
[9] *AndFTP*. [Online]. Available: http://www.lysesoft.com/products/andftp/
[10] *WhatsApp*. [Online]. Available: http://www.whatsapp.com/
[11] *Facebook*. [Online]. Available: http://www.facebook.com/
[12] *PPSTV*. [Online]. Available: http://www.pps.tv/
[13] *DropBox*. [Online]. Available: http://www.dropbox.com/
[14] *RemoteDesktop*. [Online]. Available: http://www.xtralogic.com/rdpclient.shtml
[15] A. Gember, A. Anand, and A. Akella, "A comparative study of handheld and non-handheld traffic in campus wi-fi networks," in *Passive and Active Measurement (LNCS)*, 2011, vol. 6579, pp. 173–183.
[16] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, "Energy consumption in mobile phones: A measurement study and implications for network applications," in *Proceedings of ACM Internet Measurement Conference (IMC 2009)*, 2009.
[17] *hTCDev*. [Online]. Available: http://www.htcdev.com/