All or None? The Dilemma of Handling WiFi Broadcast Traffic in Smartphone Suspend Mode

Ge Peng, Gang Zhou, David T. Nguyen, Xin Qi
Department of Computer Science, College of William and Mary
{gpeng, gzhou, dnguyen, xqi}@cs.wm.edu

***Abstract*—Smartphones save energy by entering a low power suspend mode (<20mW) when they are idle. We find that on some smartphones, WiFi broadcast frames interrupt suspend mode and force the phone to switch to active mode (>120mW). As a result, power consumption increases dramatically. To improve energy efficiency, some phones employ a hardware broadcast filter in the WiFi driver. All UDP broadcast frames other than Multicast DNS frames are blocked, thus none is received by upper layer applications. So, we have a dilemma of handling WiFi broadcast traffic during smartphone suspend mode: either receive all of them suffering high power consumption, or receive none of them sacrificing functionalities. In this paper, we propose Software Broadcast Filter (SBF) to address the dilemma. SBF is smarter than the hardware broadcast filter as it only filters out useless broadcast frames and does not impair functionalities of applications. SBF is also more energy efficient than the "receive all" method. Our trace driven evaluation shows that SBF can save up to 52% energy consumption than the "receive all" method.**

I. Introduction

Smartphones spend a large amount of time in a state where they are not actively used. This state is usually referred to as *suspend* or *sleep* mode. In this mode, the screen is off and the processor is idle, so the phone consumes very little power. For example, power consumption of Nexus One is 11 mW in *suspend* mode while it is above 120 mW in *active* mode. By turning smartphones into suspend mode while they are not in use, considerable energy can be saved.

However, incoming WiFi traffic interrupts a smartphone's suspend mode and triggers the switch to the high power active mode. One example is app notification when screen is off. Another example, which is often overlooked, is WiFi broadcast traffic. On some smartphones, such as Nexus One, WiFi driver wakes up the whole system upon receiving a WiFi broadcast frame during suspend mode. Moreover, in order to allow enough time to process the frame and possible following transmission events, WiFi driver requires a wake lock [1] of one second. The phone stays in active mode until the wake lock expires. As a result, battery drains fast even when a user is doing nothing on the smartphone. Many users have been complaining about this problem [2] [3] [4].

Actually, WLAN is not designed for smartphones at the beginning. Although WiFi broadcast frames are destined to the whole local area network, not all of them are useful to smartphones, e.g. WiFi broadcast frames for printer service discovery. It is energy inefficient to wake up the whole system and stay awake just because of these useless background broadcast frames. Smartphones have very limited battery life, so it is important to handle WiFi broadcast traffic in an energy efficient way. To improve energy efficiency, some smartphones, such as Galaxy Nexus and Galaxy S4, receive no broadcast frames except ARP and Multicast DNS frames when they are in suspend mode. With this policy, higher energy efficiency is achieved. However, this impairs the functionalities as applications can not receive any broadcast frame during suspend mode. Broadcast traffic is pervasive and important in modern networks. Many network protocols rely on broadcast to perform correctly or effectively, such as ARP, DHCP, and DNS. Some system services employ broadcast packets for resource discovery, such as NetBIOS Name Resolution. Applications also embrace broadcast packets to communicate with neighbors, such as LAN sync feature of Dropbox [5], neighbor discovery of Spotify [6], and crowdsourcing based content sharing applications [7] [8]. Failure to receive these broadcast frames results in misfunction of system services or user applications. Complaints regarding this issue [9] [10] [11] [12] have also been posted in many technical forums.

Whether to receive a broadcast frame or not? It is difficult to tell because WiFi driver has no information about what broadcast frames are needed by system services and user applications. This leads to the dilemma of dealing with WiFi broadcast traffic on modern smartphones during suspend mode: receive all and suffer high power consumption, or receive none and sacrifice functionalities. In this paper, we address the dilemma by designing a flexible packet filter that enables fine-grained policies to handle WiFi broadcast frames. Specifically, we make two contributions.

- Through measurements and analysis, we investigate how existing smartphones deal with broadcast traffic when in suspend mode. Four smartphones are used for the study: HTC Hero, Nexus One, Galaxy Nexus, and Galaxy S4. Though power consumption and functionality analysis, we reveal the problem of existing solutions, which we refer to as the dilemma of handling WiFi broadcast traffic during a smartphone's suspend mode.
- We propose Software Broadcast Filter (SBF) to address the dilemma, and compare it with solutions on modern smartphones. Through energy modeling and trace driven performance evaluation, we demonstrate the performance of the proposed method. Compared to the "receive none" solution, SBF does not impair functionalities of smartphone applications. Compared to the "receive all" solu-

978-1-4799-8381-0/15/$31.00 ©2015 IEEE

tion, SBF reduces the power consumption by up to 52%.

In literature, existing research mainly focuses on designing energy efficient broadcast/multicast protocols for wireless networks [13] [14] or reducing energy consumption of WiFi unicast traffic when a smartphone is in active mode [15] [16] [17]. However, we are different in that we study the impact of WiFi broadcast traffic when a smartphone is in suspend mode. This problem deserves attention because: (1) Broadcast traffic has a broad impact as it affects all nodes in a network. (2) Broadcast traffic is passive and typically arrives unexpectedly. To the best of our knowledge, we are the first to present measurements and analysis of different ways to deal with WiFi broadcast traffic during smartphone suspend mode.

The rest of the paper is organized as follows. First, in Section II we investigate how existing smartphones deal with WiFi broadcast traffic during suspend mode, and reveal a dilemma that existing solutions face with experimental study. Based on our observations, in Section III, we propose a solution to address the dilemma and model its energy saving. Then, in Section IV, we demonstrate the performance through trace driven performance evaluation. Finally, we discuss related works in Section V and conclude this paper in VI.

II. Revealing The Dilemma With Experiments

To reveal the dilemma, we first introduce existing solutions on four modern smartphones. Then, we carry out experiments to show how they perform in term of functionality and energy efficiency when WiFi broadcast traffic exists.

A. Understanding Existing Solutions on Modern Smartphones

To investigate how modern smartphones handle WiFi broadcast frames when in suspend mode, we analyze WiFi drivers of four commercial smartphones listed in Table I. Note that some 802.11 control and management frames are also broadcast, such as beacon frames. However, we only focus on data frames as this is the part of traffic that we can leverage. Also, we focus on MAC layer broadcast since we study behaviors of WiFi driver. In IP layer, it can be either unicast address or broadcast/multicast address. In this paper, by (UDP/ARP) broadcast frames/traffic we simply mean WiFi broadcast data frames/traffic (with UDP/ ARP data).

TABLE I: Devices used for analysis

device	Android version	kernel version	WiFi driver
HTC Hero	2.3.7	2.6.29	wlan.ko
Nexus One	2.3.7	2.6.37	bcm4329.ko
Galaxy Nexus	4.2.1	3.0.31	bcmdhd.ko
Galaxy S4	4.2.2	3.4.0	bcmdhd.ko

HTC Hero On this phone, WiFi driver receives all broadcast frames and passes them to system network stack. When a broadcast frame arrives during suspend mode, the system wakes up to process the frame. At the same time, WiFi driver requires a wake lock [1] of one second, which allows enough time to process the current frame and any following transmission events.

Nexus One This phone is equipped with ARP offload [18], which enables a network adapter to respond to ARP requests without waking up the system. For other broadcast frames, it employs the same method as on HTC Hero: waking up (resuming), staying in active state for one second, and then going back to suspend mode. WiFi broadcast frames are usually small. It will not take too much energy for the radio to receive such frames. Figure 1 shows the power consumption when a Nexus One phone wakes up to receive a WiFi broadcast frame. Although the energy cost for the phone to resume and go back to suspend is not negligible, we find that the main part of energy is consumed during the one second wake lock time triggered by the broadcast frame.

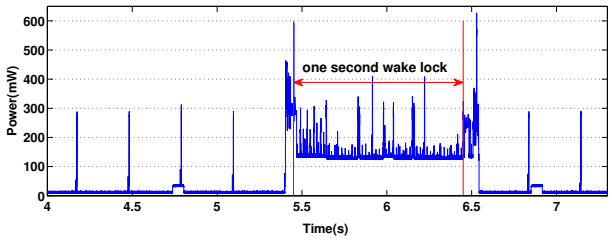

Fig. 1: Power consumption when waking up to receive a WiFi broadcast frame (measured on Nexus One with Monsoon Power Monitor[19])

Galaxy Nexus and S4 The same as on Nexus One, these two phones are also equipped with ARP offload. In addition, they enable a hardware broadcast filter. This filter blocks all UDP broadcast frames with the only exception of Multicast DNS (MDNS) frames. As a result, no UDP broadcast frames other than MDNS frames are received by the system when the phone is in suspend mode.

B. Real World WiFi Broadcast Traffic Analysis

We collect traces to see how WiFi broadcast traffic looks like in real world. Traces are collected in four locations: a corridor inside a teaching building, a college library, an office in a Computer Science Department, and an off-campus Starbucks store. For each location, we capture all broadcast frames under an AP for 30 mins during peek time.

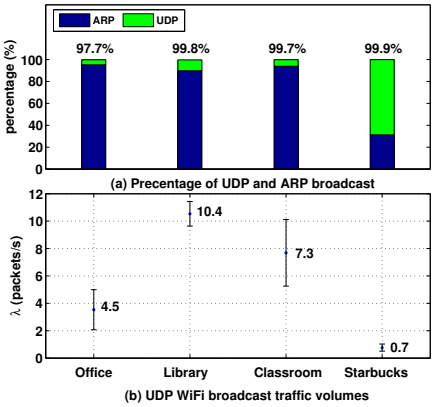

Fig. 2: Statistics of four broadcast traffic traces

TABLE II: UDP ports used by WiFi broadcast frames in traces

index	UDP port	function	index	UDP port	function
1	53	-unknown	17	6120	-unknown
2	67	DHCP bootps	18	6646	McAfee Shared Service Host, McAfee Integrated Security Platform
3	68	DHCP bootpc	19	8611	-unknown
4	137	Netbios-ns	20	8612	Canon BJNP Port 2, EMC2 (Legato) Networker or Sun Solcitice Backup, QuickTime Streaming Server
5	138	Netbios-dgm	21	9164	apani5, EMC2 (Legato) Networker or Sun Solcitice Backup, QuickTime Streaming Server
6	161	Simple Network Management Protocol (SNMP)	22	9200	-unknown
7	177	X Display Manager Control Protocol	23	9956	Alljoyn Name Service, QuickTime Streaming Server
8	631	Common Unix Printing System, Internet Printing Protocol (IPP)	24	10007	mvs-capacity
9	1004	Mac OS X RPC-based services. Used by NetInfo, for example.	25	10019	Stage Remote Service
10	1211	Groove dpp	26	17500	Dropbox
11	1900	ssdp, Microsoft SSDP Enables discovery of UPnP devices	27	23499	-unknown
12	2222	Ethernet-IP-1, trojan	28	27036	Steam In-Home Streaming Discovery Protocol
13	2223	Rockwell-csp2, Microsoft Office OS X antipiracy network monitor	29	43440	Cisco EnergyWise Discovery and Command Flooding
14	3289	enpc	30	57621	Spotify
15	3600	Trap-daemon(text relay-answer)	31	65080	-unknown
16	5353	mdns			

We calculate percentages of UDP or ARP broadcast traffic as numbers of UDP or ARP broadcast frames divided by the total number of WiFi broadcast frames. As shown in Figure 2(a), UDP and ARP broadcast frames account for more than 99% of WiFi broadcast traffic in all four scenarios.

We split the 30 mins traces into 5-min slices and calculate UDP broadcast traffic volume in terms of frame arrival rate for each slice. Figure 2(b) shows the mean and standard deviation of frame rate for each trace. The average UDP broadcast traffic volumes are all less than 11 frames/s. We also observe that traffic volumes differ largely among these locations. The lowest is 0.7 frame/s in the off-campus Starbucks store while the highest is 10.4 frames/s in the college library.

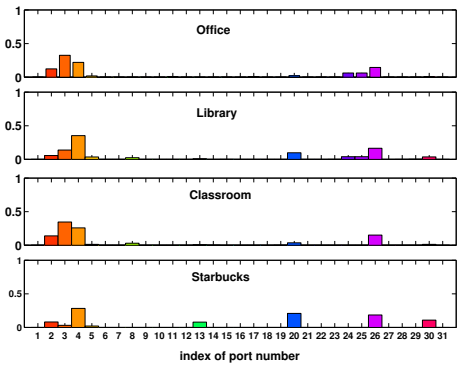

Fig. 3: UDP ports distribution (index defined in Table II)

From all four traces, we observe that 31 ports are used for UDP broadcast. We list them in Table II and show the distributions in Figure 3. Although there are 31 different UDP ports, the majority of broadcast frames lie in a small portion of them. While some of these broadcast frames are useless to a smartphone, such as Canon BJNP on Port 8612 (with index 20), some of them are useful and important to a smartphone. For example, users may keep smartphone screen off while waiting for connection from nearby devices. If the smartphone can not receive and respond to service discovery broadcast frames, such as UPnP and Steam, other devices nearby will not know presence of this smartphone. Thus, device can not be connected and service can not be used. Another interesting finding during our experiments is that LAN Sync feature of Dropbox is not included in its Android app. One reason would be that it can not work because some smartphones can not receive broadcast frames for neighbor discovery when in suspend mode. The "receive none" solution sacrifices functionalities. What's worse, without system support to receive broadcast traffic during suspend mode, developers will be pushed to abuse wake lock to prevent smartphones from suspending, so as to ensure reception of broadcast packets.

C. Power Impact Measurements

To have a better understanding of the impact of WiFi broadcast traffic on smartphone power consumption in suspend mode, we carry out experiments to show how the power consumption changes with different broadcast traffic volumes when smartphones are in suspend mode. As already shown in Figure 2(a), real word WiFi broadcast traffic mainly consists of UDP and ARP broadcast frames. Therefore, we measure the impact of UDP and ARP broadcast frames on power consumption of smartphone suspend mode respectively.

Setup For the experiments, a private AP is set up to control the background WiFi broadcast traffic volume. The traffic generator, which is a desktop, sends out UDP or ARP broadcast packets following a Poisson distribution [20]. Payloads of all broadcast packets are fixed to 50 bytes. We adjust the traffic volume by varying the value of arrival rate λ for the Poisson distribution. When $\lambda = 0$, there is no WiFi broadcast traffic. We suppress all outgoing application traffic in order to eliminate noise of transmission events. We keep WiFi connected and screen off, then measure power consumption of the whole phone with Monsoon power monitor [19], as shown in Figure 4. Each measurement lasts 5 mins and each data point is the average value of 5 repeated measurements.

Power Impact of UDP Broadcast Since we only consider background WiFi broadcast traffic, we choose a UDP port number that is not listened to by the phone. To measure the broadcast impact, we first measure the average power

Fig. 4: Experiment setup

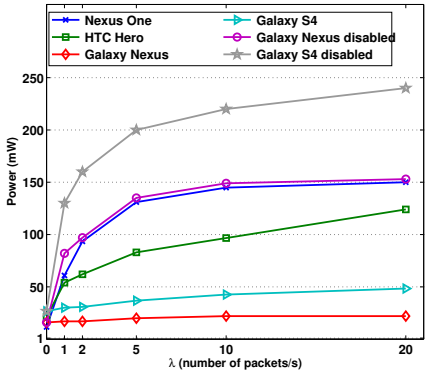

Fig. 5: Power impact of UDP broadcast

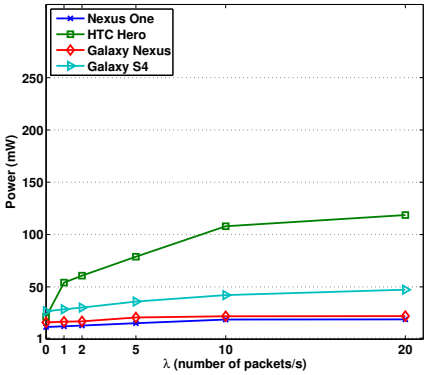

Fig. 6: Power impact of ARP broadcast

consumption of the whole phone with screen off and WiFi connected but no traffic (denoted as E_0). Then, we measure the average power consumption (denoted as E_1) of the whole phone with broadcast traffic added. The broadcast impact in terms of energy is then calculated as $E_1 - E_0$. All power consumptions are measured with Monsoon power monitor.

Figure 5 shows the results of the aforementioned four phones. As we can see, power consumptions of Galaxy Nexus and Galaxy S4 do not increase too much because these two phones receive no UDP broadcast frames during suspend mode. In contrast, HTC Hero and Nexus One receive all UDP broadcast frames. Thus, power consumptions of these two phones increase dramatically as UDP broadcast sending rate increases. Power consumptions are less than 25mW for these two phones when there is no UDP broadcast traffic. They rise above 50mW when there is only one UDP broadcast packet per second. Similar trends are also observed for Galaxy Nexus and Galaxy S4 after disabling the hardware broadcast filter, which are indicated by the curves named "Galaxy Nexus disabled" and "Galaxy S4 disabled." Additionally, for all four phones, power consumption increase is slowed down when the UDP broadcast sending rate exceeds 10 packets/s. This is because the smartphones already spend most of time in the high power active mode when there are 10 broadcast packets per second. Further increase of WiFi broadcast traffic volume would not obviously increase the portion of time in active mode.

Power Impact of ARP Broadcast In all ARP broadcast packets, the IP address to be resolved does not belong to the smartphone, as our concern is energy consumed by useless broadcast frames. As shown in Figure 6, for HTC Hero, increase of power consumption under ARP broadcast traffic is similar to that under UDP broadcast traffic. However, ARP broadcast traffic is observed to have little impact on the other three phones. For example, power consumption of Nexus One increases by less than 8mW when we increase the ARP broadcast traffic from 1 to 20 packets/s. From our analysis in the previous section, we learn that the reason is ARP offload. As observed, ARP offload is efficient enough to deal with ARP broadcast traffic. Therefore, in the rest of this paper, we target at UDP broadcast traffic.

III. Software Broadcast Filter Design and Energy Saving Analysis

As we have demonstrated, current solutions of receiving all or no broadcast frames sacrifice either functionalities or battery life of a smartphone. To address the dilemma, we design a flexible and fine-grained Software Broadcast Filter (SBF). To demonstrate the energy efficiency of SBF, we first characterize the energy consumption when a smartphone system wakes up to receive a broadcast frame. Then, we get the energy saving of SBF by modeling the energy consumptions of both SBF and "receive all" method.

A. SBF Design

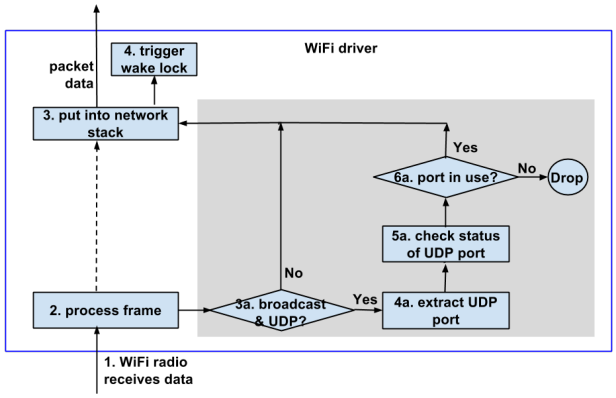

Fig. 7: SBF work flow inside WiFi driver

Figure 7 shows the work flow of SBF inside WiFi driver. Actions outside the shaded rectangle are logicals of the original WiFi driver. Actions inside the shaded rectangle are logicals of SBF. All actions are numbered in order along the work flow. For every UDP broadcast frame received by the WiFi radio, SBF extracts the UDP port number and checks with the system whether the UDP port is listened to or not (Linux kernel maintains a hash table for all UDP port numbers currently in use). If the UDP port is not listened to, this is a useless broadcast frame. SBF simply drops it without requiring a wake lock; otherwise, SBF passes the frame and lets the WiFi driver continue with the processing.

Compared to the "receive none" hardware broadcast filter, SBF is smarter in that it blocks all useless broadcast frames

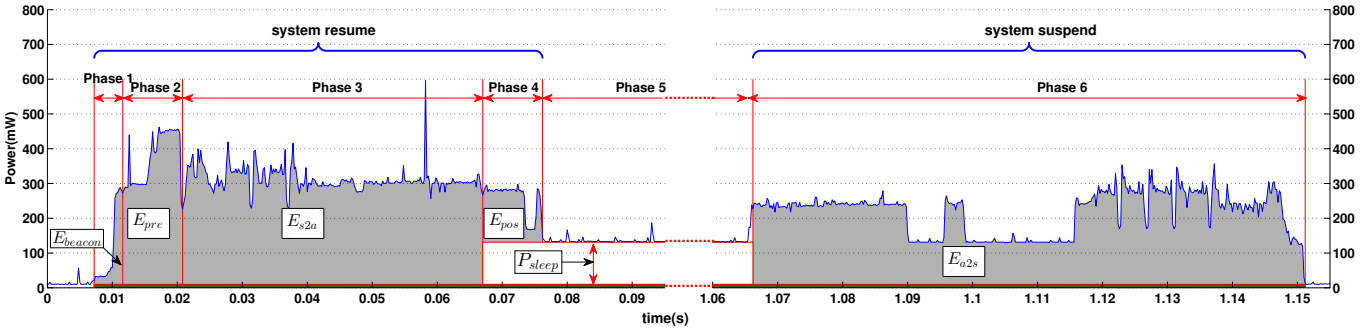

Fig. 8: power consumption during system resume and suspend (measured on Nexus One phone)

but lets the useful ones in. Thus, SBF does not impair the functionality. To analyze energy efficiency of SBF, we first build an energy model based on power profiles of Nexus One and Galaxy S4 phone. Then, we compare power consumption of SBF with that of the default "receive all" WiFi driver, based on trace driven simulation.

B. Energy Characterization

In Figure 1, we have already seen a typical process of receiving a WiFi broadcast frame during a smartphone's suspend mode: waking up from suspend, receiving broadcast frames, keeping awake for around one second if no traffic follows, and then going back to suspend. We zoom in the process and show a close look of resume part and suspend part in Figure 8.

As marked in Figure 8, there are mainly 6 phases during the whole process. At first, the phone is in suspend mode with very low power consumption ($\sim$11mW). Then, the smartphone enters the following phases one by one.

- **Phase 1 - beacon** This phase is the beginning of a Delivery Traffic Indication Message (DTIM) interval. During this phase, WiFi radio wakes up to receive the beacon frame carrying broadcast traffic information. If the beacon frame indicates that there is no frame buffered at the AP, the smartphone stays at suspend state. Otherwise, WiFi radio continues to receive data and enters Phase 2. Energy consumption during this phase is the energy consumed to receive the beacon frame E_{beacon}.
- **Phase 2 - pre-resume** This is the pre-resume phase. During this phase, WiFi radio receives the broadcast frame and sends an interrupt to the kernel. This triggers the system resume. Energy consumption during this phase is denoted as E_{pre}.
- **Phase 3 - resume** The main task of this phase is system and device resume. At the end of this phase, WiFi driver processes the broadcast frame and starts the wake lock timer which expires in one second. Energy consumed during the whole phase is denoted as E_{s2a}.
- **Phase 4 - post-resume** This is the post-resume phase. After this phase, if there are no more tasks to do and no more incoming WiFi data frames, the system becomes idle. However, as the wake lock timer is active, the system stays at active state until the timer expires. We calculate energy consumption of this phase E_{pos} as the extra power consumption when compared to power consumption during system idle.
- **Phase 5 - wake-lock** During this phase, if there are more WiFi data frames coming in, WiFi driver can process them immediately as the system is in active state during the whole phase. At the same time, new incoming data frames will update the wake lock timer to be one second. If there are no more data frames, the smartphone goes back to suspend state after the wake lock timer expires. The average power consumption of system idle during this phase is P_{sleep}. This is the phase that SBF tries to avoid or shorten. At the end of the post-resume phase or anytime during this phase, if SBF finds out that the broadcast data frame received is not listened to by any application and the "more data" bit in the frame header is not set (no more broadcast frames buffered at the AP), it goes directly to Phase 6.
- **Phase 6 - suspend** This is the phase when the system transits from active state to suspend state. We denote the energy consumption as E_{a2s}.

During all phases, the small dark space right above the x-axis in Figure 8 is the average power consumption when the system is in suspend mode, denoted as $P_{suspend}$ ($\sim$11mW).

Energy consumption of handling WiFi broadcast frames during a smartphone's suspend mode can be divided into three aspects. (1) The first aspect E_1 is energy consumed by WiFi radio to receive WiFi frames, including idle listening, data transmission, and frame processing. (2) The second aspect E_2 is energy consumed by system state transfers, including transitions from suspend to active and transitions from active to suspend. (3) The third aspect E_3 is energy consumed in system idle state due to WiFi wake lock.

SBF is a software method. It does not stop WiFi radio from receiving any broadcast frame. So, SBF does not impact energy consumed by the first aspect. From Figure 1, we see this part of energy is not dominant when broadcast traffic is sparse. SBF increases energy consumption of the second aspect because SBF puts smartphone into suspend mode more aggressively than the "receive-all" solution. With SBF, the chance that a broadcast data frame comes in when the system is in suspend mode becomes higher. As a result, the frequency of system state transfer increases. This is the overhead of SBF. However, SBF reduces energy consumed in the third

aspect because SBF reduces the time that the system spends at idle/active state after receiving a broadcast frame. During a smartphone's suspend mode, energy reduction of SBF in the third aspect is usually larger than the energy increase in the second aspect, which is why SBF saves energy. Later, in our evaluation results (Figure 9), we show how much energy is consumed in these three aspects respectively.

C. Energy Saving Modeling

Suppose that a smartphone receives n UDP broadcast frames during m beacon intervals. The i_{th} broadcast frame arrives at time t_i ($t_i > t_{i-1}$ for all $1 \leq i \leq n$) during beacon interval b_i ($1 \leq b_i \leq m$). The frame length is L_i and WiFi data rate is r_i. Beacon interval is τ_b, and it is typically configured to be 100ms in real world WiFi networks. DTIM interval is set to 1, which means Delivery Traffic Indication Messages are sent with every beacon. WiFi wake lock timer length is τ_w, which is one second on the phones we used. In order to model energy saving of SBF, we need to calculate the following parameters for each frame F_i: system state when the broadcast frame arrives $s(i)$ (1 means suspend and 0 means active), start time of wake lock timer $tw(i)$, and wake lock effective time length $T_{wl}(i)$. Without loss of generality, and to simplify the model, we assume the first beacon interval starts at time 0 and the initial state of the smartphone system is suspend, which is

$$s^a(1) = s^b(1) = 1$$
$$tw^a(1) = tw^b(1) = T_{beacon} + T_{pre} + T_{s2a}$$

T_{beacon}, T_{pre}, and T_{s2a} are time lengths of the beacon phase, pre-resume phase, and resume phase, respectively. To differentiate variables under different methods, we use superscript 'a' for variables under the "receive all" method and superscript 'b' for variables under SBF. Based on these two initial values, we can calculate the corresponding parameters for all following $n-1$ frames under the "receive all" method and SBF, respectively.

Energy Consumption of "receive all" If a frame i arrives after the suspend phase of frame $i-1$, then the system state upon frame arrival is suspend mode. Otherwise, the system is in active mode.

$$s^a(i) = \begin{cases} 0 & \text{, if } t_i \leq tw^a(i-1) + \tau_w + T_{a2s} \\ 1 & \text{, otherwise} \end{cases} \quad (1)$$

If a frame arrives *during* the suspend phase of the previous frame, it interrupts the suspend process. We assume that suspend energy cost is evenly distributed across the suspend phase. Once a suspend process is interrupted, system transits back to active mode immediately without extra transition energy consumption. If $s^a(i) = 1$ for a frame i, then the system needs to transit from suspend mode to active mode to process the frame. So, $s^a(i)$ can also be used to indicate whether a broadcast frame triggers the system resume or not.

Wake lock timer for the $i_{th}(2 \leq i \leq n)$ broadcast frame starts at time

$$tw^a(i) = \begin{cases} (b_i - 1) * \tau_b + T_{beacon} + T_{pre} + T_{s2a} & \\ \qquad\qquad\qquad\qquad\qquad \text{if } s^a(i) = 1 & (2) \\ t_i + L_i/r_i \qquad\qquad\qquad \text{if } s^a(i) = 0 & \end{cases}$$

Wake lock effective time length for the $i_{th}(1 \leq i \leq n-1)$ broadcast frame is

$$T^a_{wl}(i) = min\{\tau_w, max\{0, t_{i+1} - tw^a(i)\}\} \quad (3)$$

With the above three variables, we calculate system state transfer energy consumption of "receive all" method as

$$E^a_2 = (E_{pre} + E_{s2a} + E_{pos} + E_{a2s}) * \sum_{i=1}^{n} s^a(i) + E^a_{is} \quad (4)$$

where E^a_{is} is the energy consumed by interrupted suspends.

$$E^a_{is} = \frac{E_{a2s}}{T_{a2s}} * \sum_{i=2}^{n} T^a_{is}(i) \quad (5)$$

with

$$T^a_{is}(i) = \begin{cases} t_i - tw^a(i-1) - T^a_{wl}(i-1) & \\ \quad \text{, if } 0 < t_i - tw^a(i-1) - T^a_{wl}(i-1) < T_{a2s} & \\ 0 \text{ , otherwise} & \end{cases} \quad (6)$$

Then, the total energy consumed by the "receive all" method is calculated as

$$E^a = E^a_1 + E^a_2 + E^a_3 \quad (7)$$

where E^a_2 is already shown in Equation (4) and

$$E^a_1 = P_{idle} * T_{idle} + P_r * \sum_{i=1}^{n} \frac{L_i}{r_i} + E_{fp} * N_b \quad (8)$$

$$E^a_3 = P_{sleep} * \sum_{i=1}^{n} T^a_{wl}(i) \quad (9)$$

In Equation (8), P_{idle} is WiFi idle listening power consumption. P_r is the power consumption of WiFi when WiFi radio is receiving a frame. Since time to process a frame is very short, we assume that WiFi groups the processing of all data frames received during the same beacon interval. So, there is only a one-time frame processing energy cost during a beacon interval, denoted as E_{fp}. Also, we assume E_{fp} is constant across all beacon intervals. N_b is the number of beacon intervals with data frame(s). It is calculated as

$$N_b = \mid \{i \mid \exists b_j = i \wedge 1 \leq i \leq m \wedge 1 \leq j \leq n\} \mid \quad (10)$$

T_{idle} is the total time that WiFi radio spends at idle listening before data transmission. Considering the total idle listening time during a beacon interval b_i, it is the time offset of the last frame arrival event during the current beacon interval. Thus

$$T_{idle} = \sum_{i=1}^{m} [max\{t_j \mid b_j = i\} - (b_i - 1) * \tau_b] \quad (11)$$

Energy Consumption of SBF When SBF operates in a WiFi driver, the system state upon frame arrival is

$$s^b(i) = \begin{cases} 0, \text{ if } t_i < tw^b(i-1) + T^b_{wl}(i-1) + T_{a2s} \\ 1, \text{ otherwise} \end{cases} \quad (12)$$

Also, we have wake lock start time

$$tw^b(i) = \begin{cases} (b_i - 1) * \tau_b + T_{beacon} + T_{pre} + T_{s2a} \\ \qquad\qquad\qquad\qquad \text{if } s^b(i) = 1 \\ t_i + L_i / r_i \qquad\qquad \text{if } s^b(i) = 0 \end{cases} \quad (13)$$

and wake lock effective time length

$$T^b_{wl}(i) = max\{0, min(b_i * \tau_b, t_{i+1}) - tw^b(i)\} \quad (14)$$

where $d_{more}(i)$ stands for the "more data" bit in the MAC layer header of the i_{th} frame. If this bit is set, then SBF keeps the smartphone awake until the next broadcast frame or the next beacon interval, whichever comes first. Otherwise, SBF puts the smartphone into suspend state immediately.

Then, state transfer energy consumption by SBF is

$$E^b_2 = (E_{pre} + E_{s2a} + E_{pos} + E_{a2s}) * \sum_{i=1}^{n} s^b(i) + E^b_{is} \quad (15)$$

where E^b_{is} is energy consumed by interrupted suspends for SBF.

$$E^b_{is} = \frac{E_{a2s}}{T_{a2s}} * \sum_{i=2}^{n} T^b_{is}(i) \quad (16)$$

where

$$T^b_{is}(i) = \begin{cases} t_i - tw^b(i-1) - T^b_{wl}(i-1) \\ \quad \text{, if } 0 < t_i - tw^b(i-1) - T^b_{wl}(i-1) < T_{a2s} \\ 0 \text{ , otherwise} \end{cases} \quad (17)$$

Similarly, the total energy consumption of SBF is

$$E^b = E^b_1 + E^b_2 + E^b_3 \quad (18)$$

where E^b_2 is already shown in Equation (15) and

$$E^b_1 = E^a_1 \quad (19)$$

$$E^b_3 = P_{sleep} * \sum_{i=1}^{n} T^b_{wl}(i) \quad (20)$$

Energy Saving of SBF With the total energy consumption of both SBF and "receive all", we calculate energy saving percentage of SBF as

$$p = \frac{E^a - E^b}{E^a} \quad (21)$$

IV. SBF Performance Evaluation

We evaluate performance of SBF in terms of energy efficiency and delay through trace driven simulation. The traces we used are the four traces we introduced in Section II-B. With a Monsoon power meter, we measure and profile the power/energy consumption of two phones: Nexus One and Galaxy S4. The values are listed in Table III. To demonstrate the energy efficiency of SBF, we compare it to the "receive all" method and a calculated lower bound. To calculate this lower bound, we assume that SBF has the information of future frame arrival time. So, it can decide to keep the system active until the next broadcast frame when the wake lock energy consumption $E_3(i)$ is less than state transfer energy cost $E_2(i)$ for the current frame i. This also gives the upper bound of energy savings.

TABLE III: Energy profile

	E_{beacon}	E_{pre}	E_{s2a}	E_{pos}	E_{a2s}
NexusOne	0.41	2.72	13.88	1.11	17.66
S4	0.56	3.08	34.54	20.65	85.8
	E_{fp}	P_{idle}	P_{sleep}	$P_{suspend}$	P_r
NexusOne	1.022	370	125	11	530
S4	5.7	405	130	15	538
	T_{beacon}	T_{pre}	T_{s2a}	T_{pos}	T_{a2s}
NexusOne	0.0045	0.009	0.046	0.009	0.086
S4	0.0053	0.0114	0.044	0.039	0.165

energy in mJ, power in mW, time in $second$

A. Energy Saving

Energy savings of SBF are shown in Figure 9. In order to normalize energy consumptions across different traces, we translate energy consumption calculated with Equation (7) and (18) to average power consumption. Also, we divide the power consumption into three different aspects as presented in those two equations. For each trace, we plot three bars. The left bar is power consumption of the default "receive all" method. In the middle is power consumption of SBF. The right bar is oracle lower-bound power consumption. The values above the bars are power saving percentages. The upper values are power savings of SBF and the lower values are upper-bound power savings from the oracle we defined.

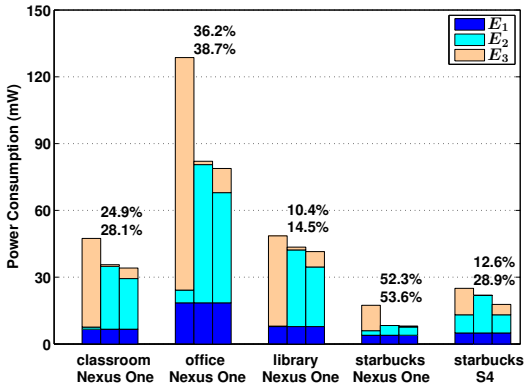

Fig. 9: Power consumptions of different methods (left bar:"receive all", middle bar: SBF, right bar: lower bound.)

Figure 9 shows that SBF saves considerable power for all traces on Nexus One phone. The largest saving is 52.3% for Starbucks trace while the smallest saving is 10.4% for library trace. And the power savings of SBF are very close to the lower-bound values from the oracle.

In order to better understand energy saving differences across different traces, we show CDF of inter arrival time of broadcast frames in each trace in Figure 10. As can be observed from the figure, Starbucks trace has obviously longer

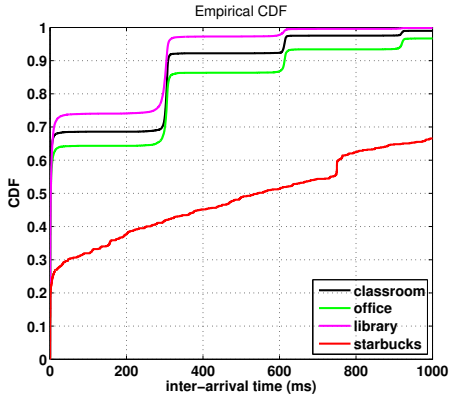

Fig. 10: CDF of inter-arrival time of broadcast frames

frame inter arrival time than the other three traces. In this case, the "receive all" method suffers because most of the smartphone's idle waiting turns out to be a waste as nothing happens. For the same case, SBF benefits the most as it reduces a lot of wake lock energy while only incurs a small amount of state transfers overhead. For the other three traces, Figure 10 shows that more than 45% of the broadcast frames have inter-arrival time less than 100ms. Which means, broadcast frames tend to arrive in batches. This is easy to understand as AP buffers these broadcast frames and sends them out together in the next DTIM interval. In contrast, S4 can only save energy for the Starbucks trace ($\sim$ 12.6%). This is because state transfer energy cost is quite high on S4 phone, as can be seen from Table III. The overhead is too heavy to be counteracted by wake lock energy reduction of SBF when the broadcast traffic is not sparse.

B. Delay Overhead

The delay overhead of SBF consists of two parts. (1) SBF takes the frame from WiFi driver, extracts the port number and looks it up in a hash table to decide whether to drop or pass the frame. So, the first part of delay is the *local processing delay*. (2) When a broadcast frame arrives during a smartphone's suspend mode, SBF needs to first wake up the system. So, the second part of delay is the *waking up latency*, which is around 60ms. Note that, this delay only impacts frames which trigger the system resume and it also incurs under the "receive all" method. Besides, SBF works during a smartphone's suspend mode where user is not delay sensitive to the traffic. Therefore, this wake up latency is acceptable for suspend mode. So, our delay evaluation here will focus on the local processing delay, as it impacts every broadcast frame received by WiFi driver.

TABLE IV: Local Delay of Software Broadcast Bilter (λ=5)

	mean (ms)	stddev
SBF	1.1464	0.0036
Receive-all	1.1343	0.0038

To measure this local processing delay, we implement the work flow shown in Figure 7 in WiFi driver of Nexus One. During the experiments, we intentionally create 100 UDP sockets on the smartphone. Then, we send 1000 UDP broadcast packets through the local area network to the smartphone. We log the time (t_1) when a frame enters step 3a in Figure 7, and the time (t_2) when the frame is received by the application. Then, we calculate the mean and standard deviation of $t_2 - t_1$ from eight repeated runs. As indicated in Table IV, the local processing is very fast. With 100 UDP ports in use, local processing delay only increases by 1.07%.

V. Related Work

A number of prior solutions have been proposed to reduce energy consumption on mobile devices. We focus on those most closely related to our work.

WiFi power consumption measurements Balasubramanian et al. [21] measure energy consumption of different components of WiFi with downloading/uploading streams. Carroll et al. [22] measure WiFi power consumption under various scenarios, such as system suspend, system idle, emailing, SMS messaging and so on. Perrucci et al. [23] measure power consumption in different stages of WiFi when the phone is downloading data. Cuervo et al. [24] also measure WiFi power consumption with different amount of downloading data. All these works focus on power consumptions consumed by application communication, while we focus on power consumption caused by background traffic.

Reducing WiFi power consumption Catnap [17] takes advantage of the bandwidth gap between wireless and wired links. They batch the time that WiFi is idle listening for data from wired network and put WiFi into sleep mode during this time. Liu et al. [25] leverage traffic prediction to exploit idle intervals as short as several hundred microseconds. All these methods reduce the time a WiFi module stays at a high power active state. However, our method reduces the time an operating system spends in high power active state when the phone is not actively used. During this time, the WiFi driver is mostly in low power sleep state. So, these work are complementary to ours.

Authors in [26] also check destination port number in WiFi frames to determine if there is a process listening on that port. They use this information to determine whether a frame is delay sensitive or not. However, transmission of broadcast frames are scheduled by AP for the whole local network. It can not be delayed for a specific client. Hence, their solution does not apply in our case.

The authors in [27] propose a solution to reduce energy consumed by 3G/LTE interface staying in active mode unnecessarily. Rozner et al. [28] try to mitigate power consumption increase when there is competitive background traffic. Bharadwaj et al. [29] study PSM timeout in WiFi driver. These two works control how wireless radio switches between active and sleep. We control how the system switches between suspend and active modes. In [30] [31], the authors study energy bugs caused by wake lock. The case they study is when wake lock is activated but is unable to be released. In our paper, the wake lock triggered by WiFi driver is due to normal behavior not a bug, as it can be released normally after the timer expires.

A similar work reducing system wakeup energy overhead caused by WiFi is done on PC [32]. They employ a peripheral

low power processor to receive all broadcast/unicast frames with less energy cost when a PC is in suspend mode. In contrast, we determine whether to take actions for a broadcast frame or not, such as activating wake lock, putting packet data to system network stack. As far as we know, we propose the first work that studies solutions for handling WiFi broadcast frames during a smartphone's suspend mode.

VI. Conclusion & Future Work

This paper studies different ways to handle WiFi broadcast traffic on modern smartphones during suspend mode. By examining WiFi drivers on four Android smartphones, we find that modern smartphones face the dilemma of handling broadcast frames during suspend mode: either receive all UDP broadcast frames suffering high power consumption or receive none UDP broadcast frame sacrificing functionalities. We analyze wireless traces under four different scenarios and show that the "receive none" solution blocks both useless and useful broadcast frames. For the "receive all" solution, we measure the impact of WiFi broadcast traffic on power consumption of smartphones in suspend mode. Results show that ARP broadcast traffic only slightly increases the power consumption due to ARP offload. However, power consumption increases dramatically as UDP traffic volume increases.

Based on these findings, we propose Software Broadcast Filter (SBF) for fine-grained UDP broadcast frame processing. Compared to "receive none" approach, SBF does not impair functionalities of smartphone applications as it only blocks useless broadcast frames. Compared to "receive all" approach, SBF saves up to 52.3% energy consumption. Meanwhile, SBF only increases the local processing delay by 1.07%.

Software broadcast filter is not perfect but opens the door for fine-grained WiFi broadcast filter research in smartphones. As future work, we plan to improve SBF in the following ways. We first plan to adapt SBF to reduce state transfer overhead. For example, SBF can decide how long to keep awake according to the current broadcast traffic volume. Second, we will combine software broadcast filter and hardware broadcast filter, switching between them according to the current context. Finally, SBF works after a WiFi radio receives a frame and the system already switches to active mode. In future, we plan to leverage a low power radio, such as Bluetooth, to wake up the smartphone only when necessary.

VII. Acknowledgement

This work was supported in part by U.S. National Science Foundation under grant CNS-1250180.

References

[1] "Android Power Management," http://elinux.org/Android_Power_Management, November 2013.
[2] "Android battery drain," http://forums.sonos.com/showthread.php?t=37497, December 2013.
[3] "How to find the root cause of your Android battery problems," http://www.reddit.com/r/Android/comments/19kq0a/how_to_find_the_root_cause_of_your_android, March 2013.
[4] "Dropbox on your home PC may be contributing to poor battery life on your phone!" http://forum.xda-developers.com/showthread.php?t=2094997, January 2013.
[5] "Dropbox LAN sync," https://www.dropbox.com/help/137/en, Accessed July 30, 2014.
[6] M. Goldmann and G. Kreitz, "Measurements on the Spotify peer-assisted music-on-demand streaming system," in *Peer-to-Peer Computing (P2P), 2011 IEEE International Conference on*, 2011, pp. 206–211.
[7] J. Teng, B. Zhang, X. Li, X. Bai, and D. Xuan, "E-shadow: Lubricating social interaction using mobile phones," in *ICDCS*, June 2011.
[8] N. Do, C. Hsu, and N. Venkatasubramanian, "CrowdMAC: A Crowd-sourcing System for Mobile Access," in *Proceedings of the 13th International Middleware Conference*, ser. Middleware, 2012, pp. 1–20.
[9] "Why are UDP packets dropped while sleeping?" https://groups.google.com/forum/#!topic/android-platform/DxlkphoLsd4, July 2012.
[10] "Udp broadcast packets not received in sleep mode," http://stackoverflow.com/questions/9363389/udp-broadcast-packets-not-received-in-sleep-mode, February 2012.
[11] "Reception of UDP packets in sleep mode," https://groups.google.com/forum/#!topic/android-platform/OpbSdp9FTmA, June 2011.
[12] "Wifi network connectivity issues and a possible fix," http://forum.xda-developers.com/nexus-4/general/wifi-network-connectivity-issues-fix-t2072930, December 2012.
[13] R. Chandra, S. Karanth, T. Moscibroda, V. Navda, J. Padhye, R. Ramjee, and L. Ravindranath, "Dircast: A practical and efficient Wi-Fi multicast system," in *IEEE ICNP*, 2009.
[14] S. Wang, S. M. Kim, Y. Liu, G. Tan, and T. He, "Corlayer: A transparent link correlation layer for energy efficient broadcast," in *ACM MobiCom*, 2013.
[15] E. Tan, L. Guo, S. Chen, and X. Zhang, "Psm-throttling: Minimizing energy consumption for bulk data communications in wlans," in *Network Protocols, 2007. ICNP 2007. IEEE International Conference on*. IEEE, 2007, pp. 123–132.
[16] R. Krashinsky and H. Balakrishnan, "Minimizing Energy for Wireless Web Access Using Bounded Slowdown," in *ACM MOBICOM*, 2002.
[17] F. R. Dogar, P. Steenkiste, and K. Papagiannaki, "Catnap: exploiting high bandwidth wireless interfaces to save energy for mobile devices," in *ACM Mobisys*, 2010.
[18] "ARP offload," http://alliedtelesis.com/manuals/at2911GP/ah1072914.html, Accessed July 30, 2014.
[19] "Monsoon solutions," http://www.msoon.com/LabEquipment/PowerMonitor/, 2014.
[20] "Traffic generation model," http://en.wikipedia.org/wiki/Traffic_generation_model, November 2013.
[21] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, "Energy consumption in mobile phones: a measurement study and implications for network applications," in *ACM IMC*, 2009.
[22] 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*, 2010, pp. 21–21.
[23] G. P. Perrucci, F. H. Fitzek, and J. Widmer, "Survey on energy consumption entities on the smartphone platform," in *Vehicular Technology Conference (VTC Spring), 2011 IEEE 73rd*, 2011, pp. 1–6.
[24] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: Making Smartphones Last Longer with Code Offload," in *ACM MobiSys*, 2010.
[25] J. Liu and L. Zhong, "Micro power management of active 802.11 interfaces," in *ACM Mobisys*, 2008.
[26] A. J. Pyles, X. Qi, G. Zhou, M. Keally, and X. Liu, "SAPSM: Smart adaptive 802.11 psm for smartphones," in *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*. ACM, 2012, pp. 11–20.
[27] S. Deng and H. Balakrishnan, "Traffic-aware techniques to reduce 3G/LTE wireless energy consumption," in *ACM CoNEXT*, 2012.
[28] E. Rozner, V. Navda, R. Ramjee, and S. Rayanchu, "NAPman: network-assisted power management for wifi devices," in *ACM Mobisys*, 2010.
[29] A. R. Bharadwaj, "Managing wifi energy in smartphones by throttling network packets," Ph.D. dissertation, Applied Sciences: School of Computing Science, Simon Fraser University, 2014.
[30] A. Pathak, A. Jindal, Y. C. Hu, and S. P. Midkiff, "What is keeping my phone awake?: characterizing and detecting no-sleep energy bugs in smartphone apps," in *ACM Mobisys*, 2012.
[31] A. Jindal, A. Pathak, Y. C. Hu, and S. Midkiff, "Hypnos: understanding and treating sleep conflicts in smartphones," in *ACM Eurosys*, 2013.
[32] Y. Agarwal, S. Hodges, R. Chandra, J. Scott, P. Bahl, and R. Gupta, "Somniloquy: Augmenting Network Interfaces to Reduce PC Energy Usage," in *NSDI*, 2009.