HIDE: AP-assisted Broadcast Traffic Management to Save Smartphone Energy

Ge Peng*, Gang Zhou*, David T. Nguyen*, Xin Qi†, Shan Lin‡
*Computer Science Department, College of William and Mary
†NSX Group, VMware Inc
‡Department of Electrical and Computer Engineering, Stony Brook University
Email: {gpeng, gzhou, dnguyen}@cs.wm.edu, xqi@email.wm.edu, shan.x.lin@stonybrook.edu

***Abstract*—WiFi is a major source of energy consumption on smartphones. Unfortunately, a non-negligible portion of the WiFi energy consumption is spent for frames that are useless to the smartphone. For example, energy is wasted to receive WiFi broadcast frames that are not needed by any smartphone application. What's worse, in order to process the broadcast frames received, a smartphone in suspend mode switches from suspend mode to high power active mode and stays there for a while. As such, additional energy is wasted to do the processing. In this paper, we design a system, namely HIDE, to reduce smartphone energy wasted on useless WiFi broadcast traffic. With our system, smartphones in suspend mode do not receive useless broadcast frames or wake up to process useless broadcast frames. Our trace-driven simulation shows that the HIDE system saves 34%-75% energy for Nexus One when 10% of the broadcast frames are useful to the smartphone. Our overhead analysis demonstrates that our system has negligible impact on network capacity and packet round-trip time.**

I. Introduction

WiFi is among the top biggest culprits for battery drain on smartphones, mainly due to two factors. First, WiFi consumes considerable amount of power on smartphones. For example, when WiFi is turned off, power consumption of Galaxy S4 is $\sim 130mW$ with system idle and screen off. When WiFi is receiving data, the power consumption adds up to $\sim 538mW$. Second, the amount of data traffic over WiFi is significant on smartphones. A report shows that WiFi accounts for 73% of total traffic on Android smartphones [1]. With mobile data offloading [2] [3], more and more smartphone traffic will flow over WiFi.

Reducing WiFi energy consumption can effectively boost smartphone battery life. Generally, energy consumed by WiFi is spent for data downloading/uploading desired by users. In some cases, unwanted (or useless) traffic may become rampant and dominate WiFi energy consumption, such as malicious traffic from attackers (e.g., denial-of-service or energy attackers) [4] [5] and background broadcast data traffic that is useless to a smartphone [6] (e.g., WiFi broadcast frames for printer service discovery). Thus, to reduce WiFi energy consumption, we seek to cut down energy waste incurred by unwanted WiFi traffic.

Existing literature mainly focuses on how to receive desired traffic in a more energy efficient way, e.g., traffic scheduling or traffic shaping [7] [8] [9]. With these methods, a client has no choice of what should be sent to it. Some other work [6] has studied how to filter out useless broadcast frames in WiFi driver at client side after they are received by the WiFi radio. In this way, useless broadcast data frames are still received by smartphones. Unnecessary energy has already been consumed to receive and process these useless data frames. What is worse, if a smartphone is in suspend mode (i.e., the system-on-chip (SOC) of the device including CPU, ROM, and the micro-controller circuits for various I/O devices are suspended [10]) when a useless frame arrives, the device still needs to switch to active mode in order to wake up the CPU and other resources to do the processing.

In this paper, we improve smartphone energy efficiency by reducing energy wasted on useless WiFi broadcast traffic[1]. Specifically, we propose to filter out useless UDP-padded broadcast frames (MAC layer WiFi broadcast data frames with UDP payload) at APs before they are received by smartphones. Thus, no energy will be wasted on smartphones to receive or process these useless broadcast frames. We focus on broadcast traffic because broadcast traffic is normal traffic that naturally exists in almost every network. In contrast, malicious unicast traffic is abnormal traffic which only exists in the targeted network. It is trivial to extend our system to incorporate useless unicast traffic. Although it is also interesting to work on other types of WiFi broadcast frames, in this paper, we focus on UDP-padded broadcast frames as they are the majority of WiFi broadcast data frames [6]. In the rest of this paper, unless specifically stated, broadcast frame/traffic means UDP-padded broadcast frame/traffic. Also, we target at smartphones in suspend mode because power consumption is very low in this state. If a data frame arrives during a smartphone's suspend mode, the smartphone needs to switch to high power active mode and stays in that mode for a while. The energy impact of useless traffic on smartphones in suspend mode is much more serious than the impact on smartphones in active mode.

However, in order to filter out useless broadcast traffic at APs, two research questions need to be answered. The first question is *how to differentiate between useful and useless broadcast traffic*. APs have no idea about what broadcast frames are needed by clients. Moreover, the definition of "useful" and "useless" is different across clients. A broadcast

[1]In this paper, we use unwanted traffic and useless traffic interchangeably.

frame which is useless to a client may be useful to another client. The second question is *how to manage useless broadcast traffic in an energy efficient way*. An AP cannot simply drop a useless broadcast frame for one client as it may be useful to other clients. Currently, the 802.11 network protocol assumes that broadcast frames are to be received by all clients. So, an AP uses only one bit in beacon frames to indicate any buffered broadcast frames to all clients. This cannot deliver client-specific notifications. Besides, communication between a client and an AP has cost. It incurs energy overhead as well as brings extra traffic to the network which may decrease network throughput.

In this paper, we answer the above two research questions. Our main idea is to enable cooperation between an AP and smartphone clients. Clients tell the AP what are needed. With the information from clients, the AP identifies useless broadcast frames for each client. Then, traffic notifications sent out within beacon frames are extended to offer one bit for each client. So, the AP can indicate to each client only useful broadcast frames. With our solution, no energy is wasted to receive useless broadcast frames. Moreover, if there are no useful frames, a client does not even need to wake up from suspend mode. Thus, our solution remarkably reduces the energy wasted on unwanted broadcast traffic. Our main contributions are:

- We design a framework, namely HIDE, working between an AP and smartphone clients to reduce smartphone energy wasted on useless broadcast traffic. In our system, broadcast frames are managed at the AP. The AP hides presence of useless broadcast frames from each client. As a result, smartphones in suspend mode do not need to receive and wake up to process these useless broadcast frames.
- We demonstrate the energy saving of our system with energy modeling and trace-driven simulation. With five broadcast traffic traces collected in five different real-world scenarios, we show that the HIDE system saves 34%-75% energy for Nexus One and 18%-78% energy for Galaxy S4 when 10% of the broadcast traffic are useful to the smartphone. Our overhead analysis demonstrates that our system has negligible impact on network capacity and packet round-trip time.

The rest of this paper is organized as follows. In Section II, we present some background on how broadcast frames are managed in current WiFi networks. Then, we propose our system in Section III. We model the energy consumption of the proposed system in Section IV and analyze its performance overhead in Section V. We evaluate our system in Section VI. After that, we introduce related work in Section VII. Finally, we draw our conclusions and discuss future work in Section VIII.

II. Background

In 802.11 networks, an AP periodically sends out a beacon frame [11]. Every client under the AP must periodically wake up the WiFi radio and receive beacon frames.

The AP buffers unicast frames for every client with WiFi radio in Power Saving (PS) mode. Notifications of unicast frames buffered at the AP are sent out in every beacon frames with a TIM (Traffic Indication Map) information element, shown in Figure 1. The notification data is encoded in the Partial Virtual Bitmap field, one bit for each client. If there are unicast frames buffered for it, the client must send a Power Save Poll (PS-Poll) control frame to retrieve each buffered frame from the AP.

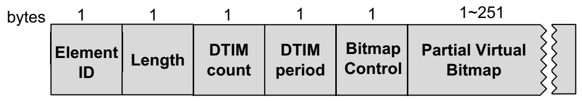


Fig. 1. Traffic Indication Map information element

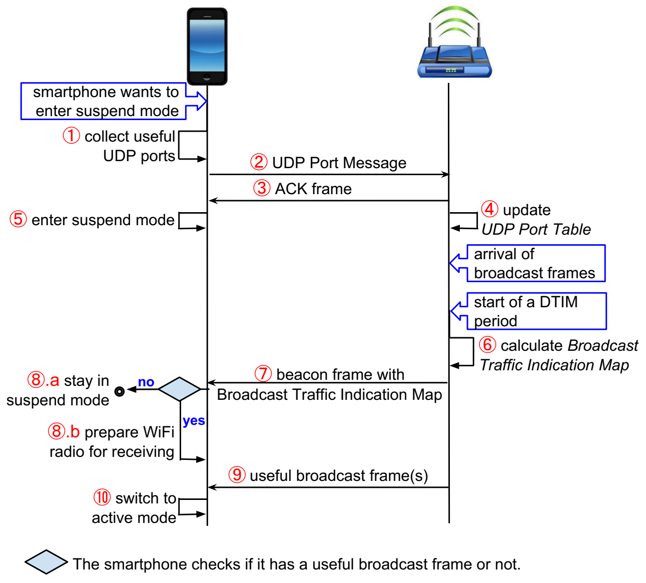
The AP also buffers all broadcast/multicast frames as long as there is one client with WiFi radio in PS mode. Notifications of buffered broadcast/multicast frames are sent out with a special type of TIM called DTIM (Delivery Traffic Indication Map). This DTIM is generated within beacon frames at a frequency specified by the DTIM period (interval). In Figure 1, DTIM period is represented in unit of beacon intervals. Typical values are $1 \sim 3$. The DTIM count field indicates how many beacons must be transmitted before receiving the next DTIM. The DTIM count is zero when we reach a DTIM. The first bit of the Bitmap Control field is used to indicate whether broadcast/multicast frames are buffered at the AP or not. If there are any broadcast/multicast frames buffered, i.e., the first bit of the Bitmap Control is set to one, every client must listen to the channel and receive the broadcast/multicast frames. After a DTIM, the AP sends the multicast/broadcast data on the channel following the normal channel access rules (CSMA/CA).

III. Proposed System

In this section, we present the proposed system. our main idea is to use UDP ports to differentiate between useless and useful UDP-padded broadcast frames. If the UDP port of a broadcast frame is opened (listened to by a process) on a client, then the AP considers this broadcast frame useful to the client; otherwise, the AP considers this broadcast frame useless to this client. Then in traffic indication, the AP hides the presence of useless broadcast frames from corresponding clients and only tells the presence of useful broadcast frames. We call the proposed system HIDE.

A. System Overview

Figure 2 shows an overview of how the system works. Every time before a smartphone enters suspend mode, it collects all UDP ports currently opened and sends them to the AP in a *UDP Port Message*. Upon receiving a *UDP Port Message*, the AP responds with an ACK frame. At the same time, the AP stores all UDP ports received from clients in a hash table (*Client UDP Port Table*) and keeps the table updated with



smartphone wants to enter suspend mode
① collect useful UDP ports
② UDP Port Message
③ ACK frame
④ update UDP Port Table
⑤ enter suspend mode
arrival of broadcast frames
start of a DTIM period
⑥ calculate Broadcast Traffic Indication Map
⑦ beacon frame with Broadcast Traffic Indication Map
⑧.a stay in suspend mode
no
yes
⑧.b prepare WiFi radio for receiving
⑨ useful broadcast frame(s)
⑩ switch to active mode
The smartphone checks if it has a useful broadcast frame or not.

Fig. 2. System Overview

the latest data from clients. After receiving the ACK frame from the AP, the client now enters into suspend mode. During suspend mode, the smartphone screen is off. The CPU, ROM, and the micro-controller circuits for various I/O devices are suspended [10]. However, the WiFi chip is still able to receive beacon frames and check if there are any frames buffered at the AP. When a DTIM period starts, the AP calculates a flag for each client based on the *Client UDP Port Table*. This flag indicates whether there are useful broadcast frames buffered for the corresponding client or not. These flags are carried in the *Broadcast Traffic Indication Map* (BTIM) information element in a beacon frame. Every client checks its exclusive bit in the BTIM information element. If this bit is not set, then no useful broadcast frames are buffered at the AP. The client stays in suspend mode as long as there are no unicast frames buffered. If the corresponding bit is set, then the client has useful broadcast frames buffered at the AP. No matter there are unicast frames buffered or not, the client needs to prepare its WiFi radio for receiving data. After data is received by the WiFi radio, the client switches to active mode, i.e., waking up the CPU and other resources, to process the frames.

In the following subsections, we present more details of the proposed system about (1) how UDP port information is sent from clients to the AP with a UDP Port Message, (2) how the AP determines whether a client has useful broadcast frames, and (3) how broadcast traffic indication flags are delivered to clients in a beacon frame.

B. UDP Port Information Synchronization

In our HIDE system, an AP uses UDP ports to differentiate *useless* and *useful* broadcast frames. This policy requires that the AP has the information of all open UDP ports on each smartphone. As this information is only available on the client itself, a client needs to send the data to the AP. The structure of this frame is shown in Figure 3. It is called UDP Port Message.

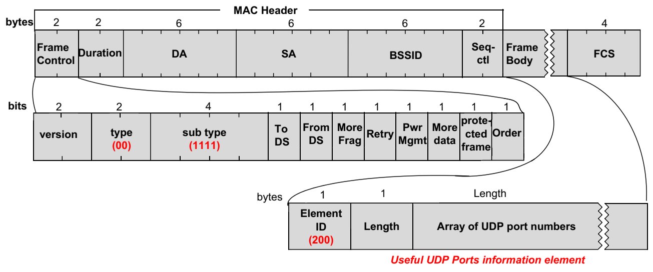


Fig. 3. Frame structure of UDP port message

A UDP Port Message is a WiFi management frame (type=00, subtype=1111) sent from a client to an AP, reporting a set of UDP ports opened on the client. To reduce the size of the message, a client only reports UDP ports associated with the source address *INADDR_ANY*. To carry the UDP port information, we add a new information element, named *Open UDP Ports* information element (as in Figure 3) to the standard 802.11 protocol. We use 200, which is reserved and unused by 802.11 protocols, as the element ID for *Open UDP Ports* information element. This information element contains an array of UDP port numbers. Each UDP port number takes 2 bytes. Upon receiving a UDP Port Message, the AP responds with an ACK frame, so that the client knows the message is successfully delivered. If an ACK frame is not received by the client, the normal retransmission operation applies to the UDP Port Message.

Each time before a client enters suspend mode, it sends a UDP port message to the AP. If there is a change made to the set of open UDP ports on a client, such as adding a new open UDP port or deleting an existing open UDP port, the system should definitely have already resumed to active mode to process such an event. Next time when the system is about to enter suspend mode, a new UDP port message will be sent to the AP with the latest UDP port information. In this way, an AP can always get the updated open UDP ports from a client.

C. Traffic Differentiation at AP

A broadcast frame may be useful to one client while being useless to another client. So, in the HIDE system, the AP maintains a broadcast flag (one bit) for every associated client. If there is any useful broadcast frame buffered for a client, the corresponding broadcast flag is set to 1; otherwise, the broadcast flag is set to 0.

Open UDP ports of all clients are stored in a hash table (*Client UDP Port Table*). With this hash table, the AP then calculates the broadcast flag for each client. The procedure is described in Algorithm 1. Right before transmission of a beacon frame representing the start of a DTIM period, the AP resets all broadcast flags to 0. Then, for every broadcast frames

Algorithm 1 Calculating broadcast flags

```
Input: broadcast frames currently buffered at the AP
       Client UDP Port Table
Output: broadcast flags for clients
 1: broadcast_flags[ ] ← {0}        // initialize the array of broadcast_flags to all 0
 2: for all broadcast frames currently buffered do
 3:     $O$ ← UDP port number from frame data
 4:     $C$ ← list of clients by Client_UDP_Port_Table lookup with key $O$
 5:     for $c_i$ in $C$ do
 6:         $k$ ← AID of $c_i$
 7:         $m \leftarrow \lceil k/8 \rceil - 1$        // octet number
 8:         $n \leftarrow k - 8 * m$        // bit number in the target octet
 9:         (the $n^{th}$ bit of broadcast_flags[m-1]) ← 1;
10:     end for
11: end for
```

currently buffered, the AP extracts the destination UDP port number from the frame data. Then, the AP looks up the hash table using the UDP port number as the key and gets a list of clients C which have this UDP port opened. After that, the AP sets the broadcast flags for all clients in C to 1.

D. Broadcast Traffic Notification

The current traffic notification uses only one bit to notify all clients of the presence of any broadcast frames. To enable fine-grained notification of buffered UDP broadcast frames, we add an information element, shown in Figure 4, in the beacon frame.

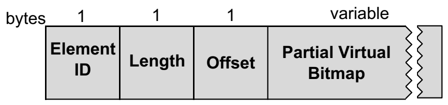


Fig. 4. Broadcast Traffic Indication Map information element

We use 201 as the element ID for our *Broadcast Traffic Indication Map* (BTIM) information element. The *Length* field indicates the total length of the subsequent fields in bytes. The *Partial Virtual Bitmap* is constructed in a similar way as in TIM information element [12] in Figure 1. The Partial Virtual Bitmap consists of the broadcast flags introduced in the previous subsection. Each bit corresponds to an Association ID (AID) of a client. For example, the 1^{st} bit is for the client with AID 1. If a bit is set to 1, then the corresponding client has useful broadcast frames; otherwise, the client does not have useful broadcast frames.

To shorten the length of this information element and reduce the protocol overhead, we do not put all flags for all clients in this bitmap. Instead, we compress the data and only put part of the flags in this field. An example is shown in Figure 5. Suppose the first N_1 (N_1 is an even number) bytes of the bitmap are all 0 and all bytes after the $(N_2)^{th}$ byte are also 0, then we can only put the $(N_1)^{th}$ to $(N_2)^{th}$ bytes in the Partial Virtual Bitmap. At the same time, we use the *Offset* field to indicate the start of the partial bitmap: *Offset* $= N_1$.

For clients who do not support this AP-assisted broadcast traffic management, they can still follow the standard 802.11 protocol: check the first bit of Bitmap Control field in the TIM information element (as introduced in the Background section) and discard our BTIM information element. So, our system works with co-existence of HIDE-enabled devices and legacy devices.

IV. Energy Modeling

In this section, we present the energy modeling for the HIDE system.

Suppose an AP sends out n UDP broadcast frames at time $\hat{t_1}$, $\hat{t_2}$, ..., $\hat{t_n}$, respectively. Also, suppose frame i is sent during beacon interval b_i with a length of L_i and a data rate of r_i. In the original system, a client receives and wakes up for every broadcast frame sent out by the AP. However, with the HIDE system, a client only receives and wakes up for broadcast frames that are useful to it. Let u_i denote whether a UDP broadcast frame i is useful to a client or not. If $u_i = 1$, then the i^{th} UDP broadcast frame is useful to the client; otherwise, the i^{th} UDP broadcast frame is useless to the client. Based on this, the UDP broadcast traffic from the AP, in the perspective of a HIDE-enabled client, is

$$n\prime = \sum_{i=1}^{n} u_i$$
$$t_i = \begin{cases} \hat{t_i} & \text{, if } u_i = 1 \\ null & \text{, if } u_i = 0 \end{cases} \quad (1)$$

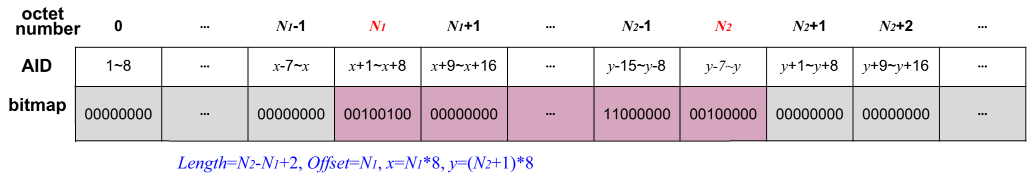
With the filtered UDP broadcast traffic, the total energy consumed by the whole system for all the n UDP broadcast frames can be calculated as

$$E = E_b + E_f + E_{wl} + E_{st} + E_o \quad (2)$$

where E_b is the energy consumed to receive all beacon frames, E_f is the energy consumed to receive all broadcast data frames, E_{st} is the energy consumed by system state transfer, E_{wl} is the energy consumption during system idle periods due to WiFi wakelocks, and E_o is the energy overhead of the HIDE system.

1) System state. Energy consumed for a UDP broadcast frame depends on the system state when the frame arrives. Thus we derive the system state first. For each UDP broadcast frame received, a wakelock of duration τ is acquired in the WiFi driver. This wakelock keeps the whole device awake and allows enough time for applications to process and respond to this frame. Also, due to the wakelock, subsequent frames can be received immediately.

If a UDP broadcast frame arrives during the wakelock of the previous frame, it renews the wakelock time and resets the *time-to-expire* to τ. If a UDP broadcast frame arrives when the system is in suspend mode, the WiFi driver needs to first wake up the operating system. If a UDP broadcast frame arrives during system resume operation, activation of the WiFi wakelock will be delayed until the resume operation is finished.



octet number	0	···	N_1-1	N_1	N_1+1	···	N_2-1	N_2	N_2+1	N_2+2	···
AID	1~8	···	x-7~x	x+1~x+8	x+9~x+16	···	y-15~y-8	y-7~y	y+1~y+8	y+9~y+16	···
bitmap	00000000	···	00000000	00100100	00000000	···	11000000	00100000	00000000	00000000	···

Length=N_2-N_1+2, *Offset*=N_1, x=N_1*8, y=(N_2+1)*8

Fig. 5. An example of the Construction of Partial Virtual Bitmap

Let $s(i)$ stands for the operating system state when frame i arrives. $s(i) = 0$ means the system is in suspend mode. $s(i) = 1$ means the system is in active state, or is resuming or suspending. Assume the wakelock for frame i starts at time $t_r(i)$, then

$$t_r(i) = \begin{cases} t_i + l_i/r_i + T_{rm} & \text{, if } s(i) = 0 \\ \max\{t_i + l_i/r_i,\ t_r(i-1)\} & \text{, otherwise} \end{cases} \tag{3}$$

where T_{rm} is the duration of system resume operation. Immediately after a system resume operation is finished, the delayed wakelocks are activated one by one in a self-renewal way. Since all these happen in a very short time, we combine them into one single wakelock. Active duration of the wakelock for frame i is

$$t_{wl}(i) = \min\{t_r(i+1) - t_r(i),\ \tau\} \tag{4}$$

Then, we calculate the system state $s(i)$. Without loss of generosity, assume $s(1) = 0$. For $2 \le i \le n$

$$s(i) = \begin{cases} 0 & \text{, if } t_i + l_i/r_i \ge t_r(i-1) + \tau + T_{sp} \\ 1 & \text{, otherwise} \end{cases} \tag{5}$$

where T_{sp} is the duration of system suspend operation.

2) Energy consumption of receiving beacon frames. The first item E_b in Eq. (2) is calculated as

$$E_b = E_b^u * \sum_{b_1 \le i \le b_n} L_i \tag{6}$$

where E_b^u is the energy consumption per byte of WiFi radio when receiving beacon frames and L_i is the length of the i^{th} beacon frame.

3) Energy consumption of receiving broadcast data frames. This second item in the right end of Eq. (2) E_f is calculated as

$$E_f = P_r * \sum_{i=1}^{n} tt(i) + P_{idle} * (\sum_{i=1}^{n} td(i) + \sum_{b_1 \le i \le b_n} tf(i)) \tag{7}$$

where P_r and P_{idle} are the power consumption of WiFi radio when receiving data and idle listening, respectively. $tt(i)$ is the transmission time of the i^{th} UDP broadcast frame, $td(i)$ is the length of time that the WiFi driver spends in idle listening state right after receiving the i^{th} UDP broadcast frame, and $tf(i)$ is the idle listening time between the i^{th} beacon frame and the first UDP broadcast frame in the i^{th} beacon interval. So,

$$tt(i) = \frac{l_i}{r_i} \tag{8}$$

$$tf(i) = \min_{j \in \{k | b_k = i\}} t_j - tb(i) \tag{9}$$

$$td(i) = \{\min\{t_{i+1}, tb(b_i + 1)\} - t_i - l_i/r_i\} * d_{more}(i) \tag{10}$$

where $d_{more}(i)$ is the 'more data' bit in the i^{th} UDP broadcast frame. If this bit is set, WiFi radio listens to the channel for future broadcast frames. $tb(i)$ is the start time of the i^{th} beacon interval and T_b is the beacon interval. Without loss of generosity, we assume $tb(1) = 0$. Then,

$$tb(i) = (i-1) * T_b \tag{11}$$

4) Energy consumption of system idle due to WiFi wakelocks. E_{wl} in Eq. (2) is calculated as

$$E_{wl} = P_{sa} * \sum_{i=1}^{n} t_{wl}(i) \tag{12}$$

where P_{sa} is the power consumption when the system is active and idle. t_{wl} is the duration of wakelock for frame i being active which is presented in Eq. (4).

5) Energy consumption of state transfers. E_{st} in Eq. (2) is calculated as this.

$$E_{st} = (E_{rm} + E_{sp}) * \sum_{i=1}^{n} [1 - s(i)] + E_{sp} * \sum_{i=2}^{n} y(i) \tag{13}$$

where E_{rm} and E_{sp} are the energy consumption of system resume and suspend operations, respectively. It may happen that a WiFi driver tries to acquire a wakelock when the system suspend operation is in execution. In this case, the system aborts the suspend operation. Let $y(i)$ denote the time portion of system in suspend operation upon arrival of frame i ($2 \le i \le n$), then

$$y(i) = \frac{\max\{0,\ t_r(i) - t_r(i-1) - t_{wl}(i-1)\} * s(i)}{T_{sp}} \tag{14}$$

6) Energy overhead. Energy overhead of our HIDE system contains two parts: energy consumed by transmission of UDP port messages E_o^1 and energy consumed by receiving extra bits in beacon frames E_o^2.

$$E_o = E_o^1 + E_o^2 \tag{15}$$

In the HIDE system, we add a Broadcast Traffic Indication Map information element in the beacon frame. So, the extra energy consumed to receive beacon frames in a HIDE system is

$$E_o^1 = E_b^u * \sum_{b_1 \le i \le b_n} L_i^b \tag{16}$$

where L_i^b is the total length of BTIM information element in beacon frame i.

In the HIDE system, a client sends out UDP Port Messages to synchronize open UDP ports with AP. This part of energy overhead, denoted as E_o^2, is calculated as

$$E_o^2 = M * P_t * \sum_i \frac{L_i^m}{r_i^m} \tag{17}$$

where M is the number of UDP Port Messages sent out by the client.

$$M = f * T_b * (b_n - b_i + 1) \tag{18}$$

In Eq. (17), P_t is the power consumption of WiFi radio when sending data. r_i^m is the data rate of the i^{th} UDP port message from a client and L_i^m is its length. From Figure 3, we see that it includes the PHY and MAC layer headers, 2 bytes of fixed fields plus a series of UDP port. Each UDP port takes 2 bytes. With N_i UDP ports in the message, we have

$$L_i^m = L_{phy} + L_{mac} + 2 + 2 * N_i \tag{19}$$

V. Network Capacity and Delay Analysis

The proposed system impacts network throughput and delay in two ways. First, in our system, AP is in charge of managing broadcast traffic. Frame processing at AP is slowed down. Consequently, packet delay is increased. Second, extra management frames (UDP Port Message) are introduced in the system. Protocol overhead is increased. Consequently, the network capacity, which is the maximum network throughput, is decreased. In this section, we quantify the impact of our system on network capacity and delay.

A. Network Capacity

In [13], the authors model the maximum network throughput that can be achieved in an 802.11 network with different numbers of nodes. We borrow their model to calculate the network capacity, denoted as S. Let Φ be the network throughput defined in [13], which is defined as the fraction of time the channel is used to successfully transmit payload bits. And let r be the average WiFi data rate (in bits/s) during transmission of payload bits. Then, the network capacity (in bits/s) of the original 802.11 network is

$$S_1 = \Phi * r \tag{20}$$

Assume that there are N clients in the network and the percent of clients with HIDE enabled is p. With our system, the total number of UDP Port Messages sent out per unit time by all clients is

$$n_u = N * p * f \tag{21}$$

where f is the frequency of sending UDP Port Messages from a client. Meanwhile, in the original network, the number of data frames transmitted per unit time is

$$n = S_1 / L \tag{22}$$

where L is the average length of payload bits in a data frame. Let L^m denote the average length of UDP Port Messages. Then, the network capacity with our HIDE system is

$$S_2 = (n - n_u * \lceil \frac{L^m}{L} \rceil) * L \tag{23}$$

Therefore, the percentage of decrease in network capacity is

$$c = 1 - S_2 / S_1 \tag{24}$$

B. Network Delay

Delay overhead of the HIDE system is mainly due to maintenance of the Client UDP Port Table and table lookup for identifying useful broadcast frames. Here, we calculate the extra network delay incurred by the HIDE system through approximate estimation.

For each UDP port message received, an AP needs to refresh the table by deleting the old ports from the hash table and inserting the new ports to the table. Assume the original round-trip time of a packet is D. Let n_o be the average number of open UDP ports in a client. With N as the total number of clients in the network, p as the percent of clients with HIDE enabled, and f as the sending rate of UDP Port Messages from a HIDE-enabled client, frame processing time at the AP will be increased by

$$t_1 = f * D * N * p * n_o * (\tau_{del} + \tau_{ins}) \tag{25}$$

where τ_{del} and τ_{ins} are the durations of a delete operation and an insert operation, respectively.

At the start of each DTIM period, for each UDP broadcast frame currently buffered, an AP needs to look up the UDP port from the hash table. Frame processing time at the AP will be further increased by

$$t_2 = n_f * \tau_{lp} \tag{26}$$

where τ_{lp} is the duration of a table lookup operation and n_f is the average number of broadcast frames buffered at AP during each DTIM period.

Then, the percentage of increase in network delay is

$$d = (t_1 + t_2) / D \tag{27}$$

Here, the delay overhead calculated is actually the upper bound, because the processing time of UDP Port Messages at the AP may overlap with part of the packet round-trip time, such as the channel access time and packet forwarding time in the backbone network. Also, a packet exchange may start and end in the middle of one DTIM period. In this case, our system does not incur the delay overhead of t_2. Thus, the delay overhead calculated by Equation (27) is the bounded network delay overhead.

VI. Evaluation

In this section, we demonstrate the performance of the proposed system, namely HIDE, by answering two questions: 1) how much energy can our HIDE system save in real-world scenarios? 2) how much does the system affect network throughput and delay?

A. Energy Efficiency

To show the energy efficiency of our system, we first present the solutions for comparison. Then, we show results of our trace-driven simulation.

1) Solutions for Comparison: To show the energy efficiency of the HIDE system, we compare its energy consumption to that of the "receive-all" method employed on modern smartphones and the lower bound energy consumption of the "client-side" solution [6].

"receive-all" solution: With the receive-all solution, the AP forwards all broadcast frames. The client receives all of these broadcast frames and activates a WiFi wakelock of one second [6] for each broadcast frame.

"client-side" solution: In the HIDE system, we manage WiFi broadcast frames at the AP side. A "client-side" solution is presented in [6]. In the client-side solution, the smartphone receives all UDP broadcast frame. Then it determines whether a broadcast frame is useful or useless. If this is a useless broadcast frame, the smartphone drops it and goes back to suspend state immediately. A "client-side" solution reduces the time that the system spends in active state due to WiFi wakelocks triggered by useless broadcast frames. However, the overhead of this solution is more frequent state transfers. We compare our method to the lower bound energy consumption of the "client-side" solution derived by the authors.

2) Trace-driven Simulation: We collect broadcast traffic traces from 5 different real-world scenarios: a classroom building, a CS department, a college library (WML), an off-campus Starbucks store, and a city public library (WRL). Each trace contains 30∼60 minutes data during peek hours. The *cdf* plots of broadcast traffic volume in the traces, i.e., number of UDP-padded broadcast frames per second, are shown in Figure 6. The average value is indicated with a black square on each curve.

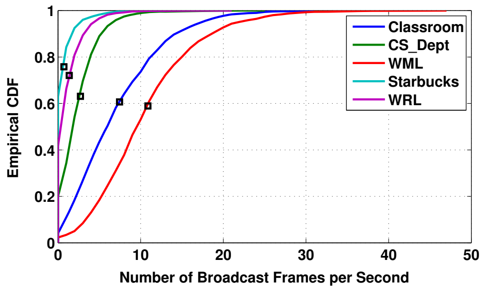


Fig. 6. Broadcast traffic volumes in traces

With these wireless traces and the energy model in Section IV, we calculate the energy consumption of different solutions through trace-driven simulation. The energy profile inputs for the model are measured with a Monsoon power monitor [14] from two phones: Nexus One and Galaxy S4. We list the values in Table I. For the HIDE system setting, we assume that the UDP port message are sent out every 10 seconds from each HIDE-enabled client with the lowest data rate of 1 Mbits/s. And, the number of UDP ports included in a UDP

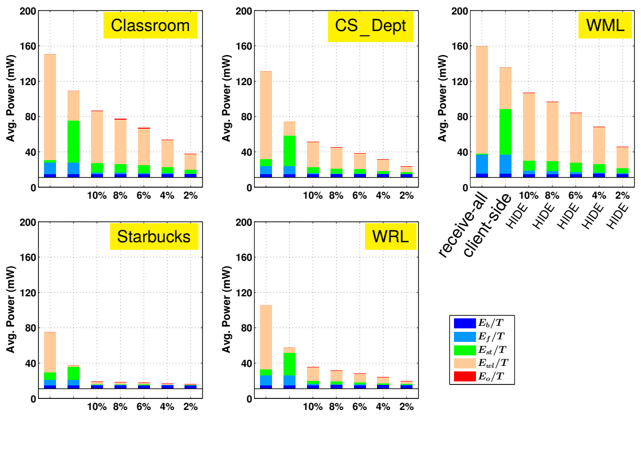


Fig. 7. Energy consumption comparison (Nexus One). Numbers along x-axis are different percentages of useful broadcast frames.

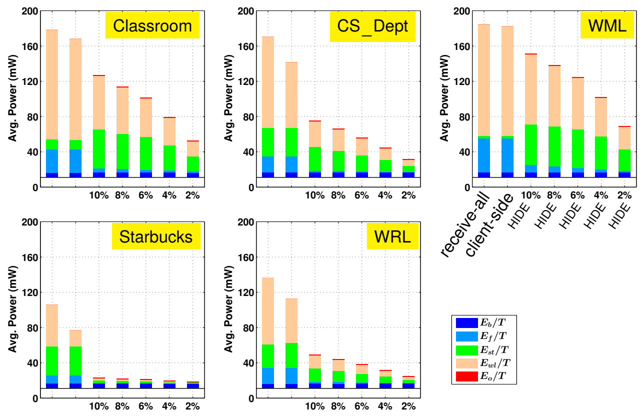


Fig. 8. Energy consumption comparison (Galaxy S4). Numbers along x-axis are different percentages of useful broadcast frames.

Port Message is set to 100. This setting is able to represent smartphones in heavy usage. Thus, they are fair enough to show the overhead of our system when compared to others.

Figure 7 and 8 shows the average power consumption of handling broadcast traffic with different solutions on Nexus One and Galaxy S4, respectively. In each sub-figure, the first bar is for the "receive-all" method and the second bar is for the "client-side" method. The last five bars are for the HIDE system with different percentages of useful broadcast frames. In order to remove the differences in duration between traces, we show the average power consumption instead of the total energy consumption. Five different colors stand for power consumed in five different aspects as introduced in Equation 2.

From Figure 7 and Figure 8, first, we see that our system saves significantly more energy than the "client-side" solution. With 10% of the broadcast frames being useful, we save 34%∼75% energy for Nexus One and 18%∼78% energy for

TABLE I
ENERGY/POWER CONSUMPTION MEASURED FROM PHONES

	τ	T_{rm}	T_{sp}	E_{rm}	E_{sp}	E_b^u	P_r	P_t	P_{idle}	P_{ss}	P_{sa}
Nexus One	1 s	46 ms	86 ms	18.26 mJ	17.66 mJ	1.25 mJ	530 mW	1200 mW	245 mW	11 mW	125 mW
S4	1 s	44 ms	165 ms	58.3 mJ	85.8 mJ	1.71 mJ	538 mW	1500 mW	275 mW	15 mW	130 mW

Galaxy S4. We save even more energy when 2% of the broadcast frames are useful: 71%-82% for Nexus One and 62%-83% for Galaxy S4. On average, HIDE:10% (the HIDE system with 10% of the broadcast frames being useful to the client) saves 23% more energy for Nexus One and 35% more energy for Galaxy S4 than the "client-side" solution. HIDE:2% saves 62% more energy on average for Nexus One and 45% more energy for Galaxy S4 than the "client-side" solution.

Second, we observe that energy savings of the HIDE system are different across traces. This is mainly because different traces have different broadcast traffic volumes. Other factors, such as frame arrival pattern, frame length, and data rate, are also causing the energy saving differences between traces. The third observation is that the energy overhead of our system, which is shown in red color, is negligible. The overhead is minimal despite that the system setting used in the evaluation represents smartphones in heavy usage.

Third, we notice that state transfer overhead on Galaxy S4 is much higher than on Nexus One. As a result, the "client-side" solution does not save much energy when the broadcast traffic is heavy, as shown in Figure 8. For example, in the classroom and college library (WML) scenarios, the "client-side" solution barely saves energy. In contrast, our system still largely reduces the average power consumption.

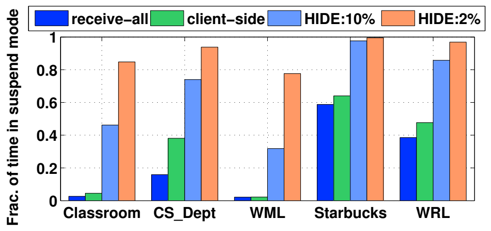


Fig. 9. Fraction of time in suspend mode for Nexus One

In order to help understand the energy savings of our method, we show the fraction of time that the device stays in suspend mode in Figure 9. HIDE:10% (HIDE:2%) means the HIDE system is used and 10% (2%) of the broadcast frames are useful. Here, we only show the results for Nexus One. Similar results are obtained for Galaxy S4. Generally, the HIDE system spends much more time in suspend mode than both the "receive-all" method and the "client-side" solution. When the broadcast traffic is heavy, such as under the classroom scenario and under the WML scenario, the device spends less than 20% of the time in suspend mode when the "receive-all" or "client-side" solution is used. However, with our method, the device spends $\geq$80% of the time in suspend mode with 2% of useful broadcast frames. One exception is that, in the CS Department scenario, the fraction of time in suspend mode for HIDE:10% is only slightly larger than that of the "client-side" solution. Referring back to Figure 7, we know that the "client-side" solution saves much less energy because it wastes a lot more energy in switching between active mode and suspend mode.

B. Impact on Network Capacity and Delay

Impact on Network Capacity. Based on the analysis in Section V-A, we calculate the percentage of decrease in network capacity with typical 802.11b network configurations as used in [15]. The parameters are listed in Table II. In addition, the sending interval of UDP Port Messages from a client is set to 10 seconds. Each UDP Port Message contains 50 UDP ports.

TABLE II
NETWORK CONFIGURATION FOR OVERHEAD ANALYSIS

min contention window	32
max contention window	1024
slot time	20 us
SIFS	10 us
DIFS	50 us
propagation delay	1 us
channel data rate	11 Mbits/s
MAC Header	224 bits
PHY preamble +header	192 bits
average data payload size	1000 bits

Figure 10 shows the results of the decrease in network capacity. We vary the total number of nodes in the network from 5 and 50. Also, we vary the percentage of nodes with HIDE enabled, denoted as p, from 5% to 75%. We made the following observations. First, the more the nodes in the network, the more the network capacity decreases. This is because the number of UDP Port Messages transmitted is linear to the number of nodes in the network. And, the original network capacity drops only slightly when the number of nodes in the network increases from 5 to 50. Second, the decrease of network capacity is negligible. With 50 nodes in the network and 75% of the nodes with HIDE enabled, the decrease of network capacity is only 0.13%. Here, we adopt the parameters from 802.11b networks. With newer 802.11 versions, the original network throughput increases largely. Thus, our system has even less network capacity overhead.

Impact on Network Delay. To measure the network delay overhead of our system, we set the percent of clients with HIDE enabled p to 50%. In addition, we set the number of

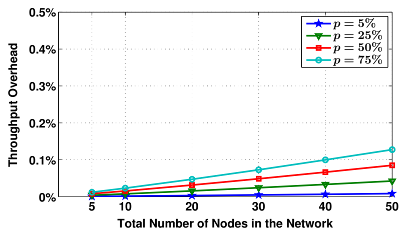


Fig. 10. Decrease in network capacity with different percents of HIDE-enabled nodes

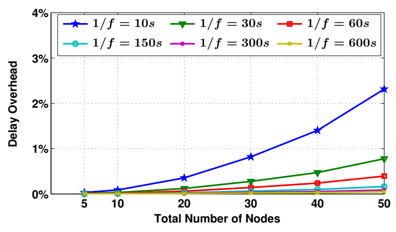


Fig. 11. Increase in network delay with different sending intervals of UDP Port Messages

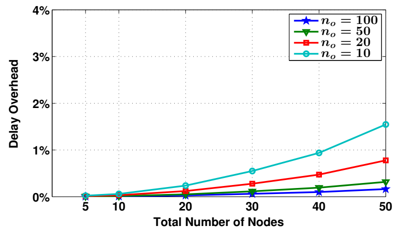


Fig. 12. Increase in network delay with different numbers of UDP ports in use

broadcast frames buffered at the AP during each DTIM period n_f to 10. Note that the n_f in the five traces we collected are all much smaller than 10. For the original network delay D, we measure the round-trip time (rtt) when connecting to a YouTube server under a deployed AP with *ping* command. In our experiments, the average rtt is 79.5ms. We use this measured rtt as the original network delay D.

To get the time durations of hash table operations, including deleting τ_{del}, inserting τ_{ins}, and lookup τ_{lp}, we implement the Client UDP Port Table on an old smartphone. We use a smartphone instead of a computer because computers have much more powerful processing capability than wireless AP/routers. The processing time measured on a computer does not reflect actual processing time on wireless APs/routers. The smartphone we use has a 1 GHz ARM processor and 512 MB memory with Android system installed. This configuration is comparable to some wireless routers in the market [16] [17]. To measure the time of operations, we first initialize the hash table with $N * 50\% * 50$ randomly generated pairs of (UDP port, Association ID). The parameter is then calculated as the mean value from 10 repeated runs of 100 deleting, or inserting, or lookup operations.

First, we fix the average number of open UDP ports in a client n_o to 50 and vary the average sending interval of UDP Port Messages. With this setup, the increase of packet delay with our HIDE system is shown in Figure 11. We observe that the more the nodes in the network, the more the packet delay increases. At the same time, the more frequently the UDP Port Messages are sent, the larger the increase is. The same as the impact on network capacity, the impact on packet delay is very small. When UDP Port Messages are sent every 10 minutes (600s), the increase of rtt is as small as 0.05%. Even when the UDP Port Messages are sent every 10 seconds, the increase is only 2.3%.

Second, we fix the sending interval of UDP Port Message to 30s and vary the average number of open UDP ports n_o in a client. The results for this configuration are shown in Figure 12. As expected, more open UDP ports means larger delay overhead. However, the overhead is less than 1.6% with 100 UDP ports in use on each HIDE-enabled client.

During our overhead analysis, we find that $t_1 \gg t_2$ in Equation (27). Meanwhile, according to Equation (25), t_1 is linear to the original network delay D. Our analysis results above actually have little dependence on the actual value of the original network delay, although we use a measured value of 79.5 ms.

VII. Related Work

The work presented in [6] is the most related work to ours. In that paper, the authors filter out useless broadcast traffic in the WiFi driver at client side after they are received by clients. With their method, energy is still wasted to receive useless broadcast frames and wake up to do the processing. In contrast, with our system, a smartphone in suspend mode does not need to receive useless broadcast frames or wake up to process the frames. Our evaluation already shows that our method saves more energy than "client-side" solutions including the work in [6].

Detecting/Filtering Unwanted Traffic. In [18] [19], the authors measure the impact of unwanted traffic on 3G networks. In [20], the authors measure the impact of unwanted link layer WiFi frames due to client association/dissociation and probe activities. In this work, we focus on WiFi broadcast frames generated by upper-layer applications with UDP payload.

The authors in [21] [5] detect data traffic sent from DDoS attackers.The authors in [22] consider null data frames from attackers as unwanted traffic and propose defense mechanisms against it. These works focus on detecting and filtering abnormal traffic from malicious nodes. However, in our work, we study WiFi broadcast frames that are normal traffic from benign nodes.

Smartphone Traffic Reduction. In [23] [24] [25], the authors propose to reduce data received by smartphones during video chatting or streaming. However, these methods target at unicast frames for a specific type of application. In this work, we consider broadcast frames that come from various applications.

In [26], the authors propose to let the server selectively send the data to a smartphone according to the smartphone's battery status. Smartphone advertising is also one source of unnecessary or unwanted traffic [27] [28]. Applications, such as Adblock [29], have been provided to block such kind of unwanted traffic. Again, all of these work study unicast traffic. We study broadcast traffic.

In [30], the authors propose to reduce general data traffic of smartphones by applying redundancy elimination at different protocol layers. Their work is orthogonal to ours.

VIII. Conclusion and Future Work

Energy is wasted on smartphones to receive broadcast frames that are useless to the smartphone and to switch from low power suspend mode to high power active mode to process these useless WiFi broadcast frames. In this work, we propose a framework, namely HIDE, to reduce energy wasted on smartphones due to useless broadcast frames, with assistance from the WiFi Access Point (AP). In the HIDE system, a client coordinates with the AP to identify useful broadcast frames. Then, traffic notifications sent out from AP only indicate useful broadcast frames that are currently buffered at the AP. The presence of useless broadcast frames is hidden by the AP from the client. As a result, a client in suspend mode does not need to receive the useless broadcast frames. Neither does it need to switch to active mode and process the useless frames.

With WiFi broadcast traces collected from 5 different real-world scenarios, we conduct trace-driven simulation with the energy model derived in the paper. The results show that our system saves 34%-75% energy for the Nexus One phone and 18%-78% for the Galaxy S4 phone when 10% of the broadcast frames are useful to the smartphone. When 2% of the broadcast frames are useful to the smartphone, our system saves 71%-82% energy for Nexus One and 62%-83% for Galaxy S4. We also analyze the performance overhead of the proposed system. The impact of the HIDE system on network capacity is less than 0.2% and the impact on packet round-trip time is no more than 2.3%.

In future, we plan to evaluate the system with more broadcast traffic traces and for more smartphones. Combining the HIDE system with the "client-side" solution is also one direction to be explored.

Acknowledgment

This work was supported in part by U.S. National Science Foundation under grants CNS-1253506 (CAREER) and IIS-1231680.

References

[1] Informa Report, "Understanding todays smartphone user," http://www.informatandm.com/wp-content/uploads/2013/06/Mobidia-ITM-June-2013.pdf, 2012.

[2] "Mobile data offloading," https://en.wikipedia.org/wiki/Mobile_data_offloading.

[3] "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update White Paper," http://tinyurl.com/mokcut3, Feb. 2015.

[4] R. Racic, D. Ma, and H. Chen, "Exploiting MMS vulnerabilities to stealthily exhaust mobile phone's battery," in *Securecomm and Workshops, 2006*. IEEE, 2006, pp. 1–10.

[5] T. Martin, M. Hsiao, D. Ha, and J. Krishnaswami, "Denial-of-service attacks on battery-powered mobile computers," in *Pervasive Computing and Communications, 2004. PerCom 2004. Proceedings of the Second IEEE Annual Conference on*. IEEE, 2004, pp. 309–318.

[6] G. Peng, G. Zhou, D. T. Nguyen, and X. Qi, "All or None? The Dilemma of Handling WiFi Broadcast Traffic in Smartphone Suspend Mode," in *IEEE INFOCOM*, 2015.

[7] A. J. Pyles, Z. Ren, G. Zhou, and X. Liu, "SiFi: exploiting VoIP silence for WiFi energy savings insmart phones," in *Proceedings of the 13th international conference on Ubiquitous computing*. ACM, 2011, pp. 325–334.

[8] 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*. ACM, 2010, pp. 91–106.

[9] J. Manweiler and R. Roy Choudhury, "Avoiding the rush hours: WiFi energy management via traffic isolation," in *Proceedings of the 9th international conference on Mobile systems, applications, and services*. ACM, 2011, pp. 253–266.

[10] A. Jindal, A. Pathak, Y. C. Hu, and S. Midkiff, "Hypnos: understanding and treating sleep conflicts in smartphones," in *Proceedings of the 8th ACM European Conference on Computer Systems*. ACM, 2013, pp. 253–266.

[11] M. Gast, *802.11 wireless networks: the definitive guide*. "O'Reilly Media, Inc", 2005.

[12] P. Roshan and J. Leary, *802.11 Wireless LAN fundamentals*. Cisco press, 2004.

[13] G. Bianchi, "Performance analysis of the IEEE 802.11 distributed coordination function," *Selected Areas in Communications, IEEE Journal on*, vol. 18, no. 3, pp. 535–547, 2000.

[14] "MonSoon Solutions," http://www.msoon.com/LabEquipment/PowerMonitor/, last acceesed in August, 2015.

[15] H. Wu, Y. Peng, K. Long, S. Cheng, and J. Ma, "Performance of reliable transport protocol over IEEE 802.11 wireless LAN: analysis and enhancement," in *IEEE INFOCOM*, 2002.

[16] "Linksys WRT1200AC," https://wikidevi.com/wiki/Linksys_WRT1200AC, last acceesed in August, 2015.

[17] "Netgear R7000," https://wikidevi.com/wiki/Netgear_R7000, last acceesed in August, 2015.

[18] F. Ricciato, E. Hasenleithner, P. Svoboda, and W. Fleischer, "On the impact of unwanted traffic onto a 3g network," in *Security, Privacy and Trust in Pervasive and Ubiquitous Computing, 2006. SecPerU 2006. Second International Workshop on*. IEEE, 2006, pp. 8–pp.

[19] I. Puustinen and J. K. Nurminen, "The effect of unwanted internet traffic on cellular phone energy consumption," in *New Technologies, Mobility and Security (NTMS), 2011 4th IFIP International Conference on*. IEEE, 2011, pp. 1–5.

[20] R. Raghavendra, E. M. Belding, K. Papagiannaki, and K. C. Almeroth, "Unwanted link layer traffic in large IEEE 802.11 wireless networks," *Mobile Computing, IEEE Transactions on*, vol. 9, no. 9, pp. 1212–1225, 2010.

[21] Y. Chen and K. Hwang, "Collaborative detection and filtering of shrew DDoS attacks using spectral analysis," *Journal of Parallel and Distributed Computing*, vol. 66, no. 9, pp. 1137–1151, 2006.

[22] W. Gu, Z. Yang, D. Xuan, W. Jia, and C. Que, "Null data frame: A double-edged sword in IEEE 802.11 WLANs," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 21, no. 7, pp. 897–910, 2010.

[23] X. Qi, Q. Yang, D. T. Nguyen, G. Zhou, and G. Peng, "LBVC: towards low-bandwidth video chat on smartphones," in *MMSys*, 2015, pp. 1–12.

[24] L. Keller, A. Le, B. Cici, H. Seferoglu, C. Fragouli, and A. Markopoulou, "MicroCast: cooperative video streaming on smartphones," in *Proceedings of the 10th international conference on Mobile systems, applications, and services (Mobisys)*. ACM, 2012, pp. 57–70.

[25] M. A. Hoque, M. Siekkinen, and J. K. Nurminen, "Using crowd-sourced viewing statistics to save energy in wireless video streaming," in *Proceedings of the 19th annual international conference on Mobile computing & networking*. ACM, 2013, pp. 377–388.

[26] M. W. Kim, D. G. Yun, J. M. Lee, and S. G. Choi, "Battery life time extension method using selective data reception on smartphone," in *Information Networking (ICOIN), 2012 International Conference on*. IEEE, 2012, pp. 468–471.

[27] A. Albasir, K. Naik, B. Plourde, and N. Goel, "Experimental study of energy and bandwidth costs of web advertisements on smartphones," in *Mobile Computing, Applications and Services (MobiCASE), 2014 6th International Conference on*. IEEE, 2014, pp. 90–97.

[28] J. Gui, S. Mcilroy, M. Nagappan, and W. G. Halfond, "Truth in Advertising: The Hidden Cost of Mobile Ads for Software Developers," in *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, vol. 1, May 2015, pp. 100–110.

[29] "Adblock Plus for Android," https://adblockplus.org/android-about, last acceesed in August, 2015.

[30] F. Qian, J. Huang, J. Erman, Z. M. Mao, S. Sen, and O. Spatscheck, "How to reduce smartphone traffic volume by 30%?" in *Passive and Active Measurement*. Springer, 2013, pp. 42–52.