Energy Efficient Wi-Fi Management for Smart Devices

Jian Li*, Jin Xiao†, Huu Nhat Minh Nguyen§, James Won-Ki Hong*, Raouf Boutaba‡

*Division of IT Convergence Engineering, POSTECH, Pohang, Korea
email: {gunine, jwkhong}@postech.ac.kr
†IBM T. J. Watson Research Center, USA
email: jinoaix@us.ibm.com
‡David R. Cheriton School of Computer Science, University of Waterloo, Canada
email: rboutaba@cs.uwaterloo.ca
§Institute of Logistics Information Technology, Pusan National University, Pusan, Korea
email: nhatminhhcmut@gmail.com

***Abstract*—Data communication is a significant component of smart device energy consumption today, and is likely to increase with the rising connectivity demands of today's mobile applications. Wi-Fi is a major supporting technology and as such, efficient energy management solutions are much needed. In this paper, we detail the Wi-Fi energy consumption characteristics and propose two energy saving schemes: packing and alignment, implementable at the application layer. Our investigations also reveal that these schemes are highly reliant on the network metric - available bandwidth, which is not readily obtainable on smartphones and existing wired solutions are ill-suited due to the dynamics of the wireless environment, as well as the energy constraint of mobile devices. Therefore, we propose a new energy efficient bandwidth measurement tool called BreezChirp. As validation, we implemented both BreezChirp and the Wi-Fi management schemes on modern Andriod smartphones and evaluated their performance through field experiments.**

***Keywords*—Energy saving, Smartphones, Management**

I. Introduction

With the rise of smart devices, mobile data communication is a rapidly growing business sector today. Anywhere anytime data connectivity unfortunately comes at a cost: increased energy consumption on a device with limited battery life. Recent study [1] have shown that energy consumption due to data communication is a significant portion of the smartphone energy consumption. And this portion is likely to increase with today's growing mobile application connectivity. From a communication technology point of view, Wi-Fi is currently a preferred carrier of mobile data because of zero cost (consumer incentive) and radio network load-shedding (provider incentive). It then follows that efficient Wi-Fi energy management is a very important practical problem for smart phones. As it stands today, a mobile device's Wi-Fi chipset comes with on-state energy management scheme in the form of dual mode switching: Power Save Mode (PSM) and Constantly Active Mode (CAM). As documented in [2], PSM mode consumes 1/5 of the power compared to CAM mode, but has very low transmission rate. The switching between PSM and CAM modes is automatically trigged based on number of packets transmitted per second. The switching delay is non-negligible and can affect user experience depending on the data intensity of the specific application. Thus far, vast majority of the energy saving solutions in literature [3][4] try to keep the device in PSM mode for as long as possible (PSM elongation) and are implemented by modifying the signaling and packet scheduling mechanism at the protocol level. These approaches have two major shortcomings: one, protocol modification deviates from standardization and is thus harder to implement and slower to gain industry adoption; two, being application agnostic hamstrings energy efficiency. For instance, today's mobile data traffic is typically a mixture of delay sensitive and delay tolerant applications, the way their communication patterns interleave to a large degree determines the energy consumption rate for which the network layer has no influence on. Thus in this paper, we take an application-centric stance to Wi-Fi energy management. First we observe the key characteristics of Wi-Fi energy consumption through experiment studies. Based on our observations, we propose two real-time application level energy saving schemes: application packing and application alignment. The key idea is that by managing the application transmission patterns based on their characteristics, we can lower energy consumption rate without sacrificing application performance and user experience.

Our investigation also reveal that these two schemes are highly reliant on timely and accurate information about available bandwidth. Unfortunately accurate and energy efficient bandwidth measurement for Wi-Fi is not readily available on smartphones currently. For management purposes, Received Signal Strength Indicator (RSSI), a physical layer attribute, is a very poor bandwidth indicator. The main obstacle is that active measurement is needed to obtain accurate bandwidth estimation, and the measurement duration required for precision in Wi-Fi environment [5][6] is prohibitive in terms of energy and communication overhead, and thus ill-suited for smart devices. We therefore propose an energy efficient bandwidth measurement tool called Battery Resource Efficient mEasurement Solution (BreezChirp). BreezChirp is accurate, energy efficient and implementable on smartphones. For validation, we implemented both BreezChirp and our real-time energy management schemes on modern smartphones and conducted

978-1-4799-0913-1/14/$31.00

performance evaluation in real field experiments. We also compare the performance of BreezChirp with pathChirp [7], a well known bandwidth measurement tool for wired network. Supported by BreezChirp, we show through field experiments running popular mobile applications, how our energy management schemes can significantly reduce Wi-Fi energy cost.

The main contributions of our work are: we studied Wi-Fi communication characteristics and proposed real-time Wi-Fi management solutions that are efficient and **application-centric**; we developed BreezChirp Wi-Fi bandwidth measurement tool that is **energy efficient** and **accurate**; our solutions are fully **implemented** on smartphones and showed good performance in real world usage scenarios.

The remainder of the paper is organized as follows. Section II presents related works on mobile device energy management, and active available bandwidth measurement techniques, and we presents two Wi-Fi energy saving schemes for mobile devices in Section III. In section IV, we study chirp more closely and present BreezChirp. Section V reports on the experiment results on BreezChirp, as well as the two energy saving schemes. Section VI concludes the paper.

II. Related work

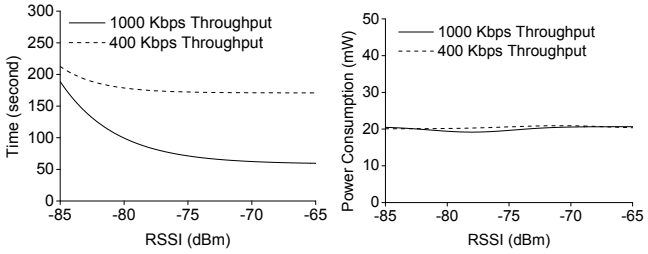
Network buffering has been proposed as an energy saving solution. In [8], the authors proposed Catnap, which exploits the bandwidth discrepancy between Access Point (AP) (low) and mobile phone (high). They observed that slow speed between AP and server force the client to be in active mode longer than necessary while the residual Wi-Fi bandwidth is high. They therefore introduced a proxy-based modification to AP which serves as an external buffer to store data from server to AP and then burst transmit them from AP to mobile. The main drawbacks of this approach are: it requires proxy on the AP, and such buffer exposes user data at unsecured public locations; it assumes that wireless channel has higher speed than wired part which is not always the case with public Wi-Fi. Other schemes consider PSM elongation at the network level. M-PSM [4] is such an enhancement in which additional power-saving opportunities are leveraged by considering not only user mobility but also traffic conditions. Unfortunately it requires network protocol modification.

Active measurement of available bandwidth in literature generally uses one of two techniques: packet dispersion and self-induced congestion. For packet dispersion technique, two or more packets are sent back-to-back to estimate capacity of a network path based on inter-packet arrival rate. pathrate [9] is the most well-known representative tool utilizing this technique. In presence of crossing traffics however, this technique becomes ineffective with high error rate. Furthermore, it introduces significant communication overhead and has long convergence time. On the other hand, self-induced congestion relies on the fact that if the probing rate exceeds the available bandwidth, then the probe packets become queued, and consequently the receiver observes elongated delay. Therefore, with varying probing rate, estimate available bandwidth can be obtained by detecting congestion at the receiver. The proposed techniques in literature generally differ in the design of the rate controller used for the probing packet train. For instance, Pathload [10] uses constant bit rate of packet train, TOPP [11] uses linearly increasing probing rate of packet pairs, while pathChirp [7] uses exponentially increasing probing rate of packet train. pathChirp has been shown to perform very well in wired networks and is relatively resource conscious compared with Pathload in that the self-induced congest state is short lived, and overall fewer number of trains are needed to obtain a good bandwidth measurement.

To date, few works exist on wireless bandwidth measurement. Wireless bandwidth estimation tool (Wbest) [5] and DietTOPP [12] are two good examples. Wbest proposed packet dispersion technique to improve accuracy and convergence time in wireless environment. However, cross traffic significantly affects their performance. DietTOPP is a simplified version of TOPP. Similar to TOPP, DietTOPP injects significant amount of probing traffic into the network. In general, measurement accuracy and energy efficiency in Wi-Fi remains an open challenge.

III. Energy Efficient Wi-Fi Management

In this section, we first investigate the key metrics which influence Wi-Fi energy consumption by performing a set of experiments, and then based on the observation, we introduce two application management schemes - application *packing* and application *alignment*, for the purpose of increasing energy efficiency. Both schemes rely on the timely and accurate measurement of available bandwidth, which is still an open problem. We address this issue with BreezChirp in Sections IV.



(a) Correlation between RSSI and time with transmitting same sized data in two data rates

(b) Correlation Between RSSI and Energy with Transmitting Data within The Same Time in Two Data Rates

Fig. 1. Correlation Between RSSI, Energy, and Time Consumption in Two Data Rates with Different Parameter Setups

A. RSSI, Application Throughput and Energy Consumption

Received Signal Strength Indicator (RSSI) is a metric that quantifies the power level present in a received radio signal. When RSSI is low, data loss and error rate increases, and to compensate, the transmitter dynamically lower the data transmission rate and therefore the overall channel capacity is reduced. To clearly understand the relation between RSSI and application throughput with respect to energy consumption, we performed a set of experiments using Samsung Galaxy S3 smartphone. PowerTutor [2] is used as the power measurement tool in our experiments.

Figure 1(a) shows the total transmission time it takes to download a fixed amount of data from a server at different throughput. It is logical that lower throughput translates to

longer time. We further understand that since the transmission is in CAM mode, the *total* power consumption due to longer transmission time is also higher. Figure 1(b) shows the energy consumption *rate* in the same experiment. Interestingly, the energy consumption rates in CAM do not differ significantly even though the transmission rate is different. Therein lies the principle of application-aware energy saving: to achieve good energy saving, we should always aim to perform data transmission at maximum throughput (which is also known as available bandwidth) whenever possible. From application point of view, what this principle means is that we want to synchronize data transmissions and burst data in CAM mode whenever possible.

B. Application Packing and Alignment Schemes

To date, vast majority of the energy saving solutions in literature manage the Wi-Fi traffic only at the network level, without understanding the application requirements and characteristics. As a result, the proposed schemes tend to be either too soft (i.e., do not provide sufficient energy saving under a realistic workload) or too hard (i.e., break user experience due to excessive delays). Therefore it is important to understand what are the critical application characteristics and how they impact and constrain the design of energy saving mechanisms.

There are several application characteristics which are highly related to energy expenditure, and among them, transmission delay sensitivity is the key factor in providing quality of user experience. By using this criterion, we can classify the applications into two general categories: delay sensitive and delay tolerant. Delay sensitive applications often require frequent user interactivity such as Instant Messaging (IM) Applications. Delay tolerant applications tolerate certain amount of delay, and include background download applications, online streaming, etc.

For delay tolerant applications, we can deliberately delay their transmission for certain amount of time (without breaking an application's user experience) in hope of bundling multiple applications to burst transmit at the same time. We term this "packing". The goal of packing is to extend the Wi-Fi module's PSM time and to achieve maximum bursting when transmitting. However, in many cases we can expect a mixture of delay sensitive and delay tolerant applications running concurrently. In this case, the transmission behavior of delay sensitive applications can be used to "cue in" packing schedules of the delay tolerant applications. By doing so, we can affix the communication of delay tolerant applications such that they are "aligned" to the communication patterns of the delay sensitive applications, and we term this "alignment". Details of the two schemes can be found in [13].

Keeping the application sensitivity characteristic in mind and using above two energy saving schemes, we design and implement an energy management application called Battery Manager (BattMan) in this paper. Figure 2 shows its architecture. BattMan is comprised of three main modules:

- **Application Classifier:** this module manages the smartphone applications based on category: delay sensitive applications and delay tolerant applications. To ease the users' job of defining input configurations, several energy saving templates are provided by default which include many popular mobile applications. In addition, the users can customize the provided energy saving templates in accordance with their preference.

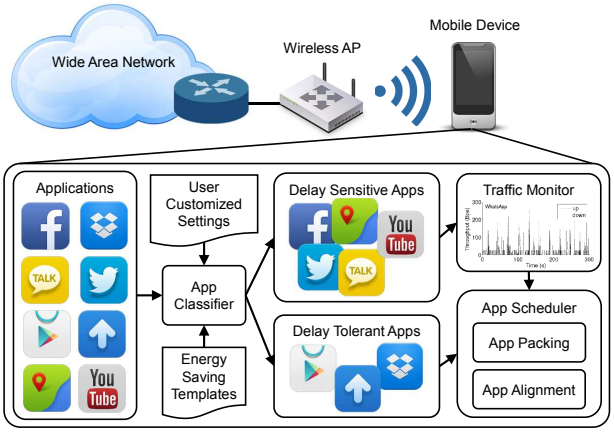


Fig. 2. System Architecture of BattMan

- **Traffic Monitor:** to realize the application packing and alignment algorithm, we need to know the application traffic pattern. Traffic monitor passively determins the applications' traffic pattern and stores it in historical profiles for later reference. Note that the traffic pattern of delay sensitive applications is determined by the users' usage pattern; therefore, it is difficult to predict the traffic pattern of delay sensitive application by referring the historical profiles even if we store them. With considering this constraint, we only store the historical profiles of delay tolerant application in this work.

- **Application Scheduler:** two data bundling schemes, named packing and alignment are implemented by the application scheduler. Application scheduler takes two input, traffic statistics and a list of delay tolerant applications. If there is no traffic information from delay sensitive applications, the scheduler simply run the packing scheme on delay tolerant applications, otherwise, it runs the alignment scheme. We adopt two mechanisms for realizing application scheduling: 1) manipulation of the process life-cycle (kernel space); and 2) traffic shaping via firewall (network space) [13]. The former scheme suspends and resums the applications (or processes) in kernel space. It works because once the process is suspended, it will no longer receive or transfer the data. As the former scheme relies on process manipulation in kernel space, it can only operate properly in the rooted mobile devices. For the unrooted devices, we can go with the latter scheme which exploits the firewall policy to shape the traffic. With the help of the latter scheme, we can easily schedule an application's trasmission by inserting the application into the blacklist of the firewall, while maintaining the others application in the whitelist of the firewall.

At the same time however, we also need to know how much we can pack, since the total amount of bandwidth is bounded. Overpacking can degrade application performance and cause unintentional congestion. Therefore, we need a method that can cheaply (energy wise) but accurately obtain the available bandwidth in smart devices. In wired networks, available

bandwidth measurement is well investigated, and there is vast literature on this. However, it remains an open problem in wireless environments especially for smart devices due to following reasons:

1) **High Wireless Channel Variation:** The wireless channel exhibits high variance caused by un-predictable channel contention and varied signal strength of wireless AP to terminal. This makes precise measurement very difficult.
2) **Timely and Accurate:** There are many mobile applications executed in a short period of time, which requires frequent data bundling to be performed. In order to bundle the data, an updated bandwidth availability is required, therefore, fine-grained bandwidth measurement is required.
3) **Large Energy Expenditure:** Existing bandwidth measurement techniques are all designed for PCs which do not suffer from energy constraint problem, while energy efficiency is of the utter importance to smart devices.

Issue one and two call for prolonged and frequent bandwidth measurements to ensure information precision and fidelity, while issue three strives for short and infrequent measurement. These lead to a logical impasse. In the next section we address these issues with BreezChirp.

IV. Chirp and BreezChirp

In this section, we discuss how BreezChirp helps to make our BattMan management solution work. BreezChirp is energy efficient by employing short lived probing chirps that works in PSM mode. Before we detail the design of BreezChirp, we first study the construction of conventional chirp and how it can be reduced to be energy efficient for smartphones.

A. Working With Chirp

The idea of chirp is first proposed by pathChirp. It uses m number of packet trains called chirps. In each chirp, N exponentially spaced probe packets reside. The relative queueing delay between chirp sender (measurement host) and receiver (measurement server) is measured based on the sequence of arrival times at the receiver, and available bandwidth is estimated accordingly. Each probe packet has identical packet size $\mathbf{P}$ bytes and the inter-packet time Δ between two consecutive packets is exponentially decreased by a spread factor γ. As Δ decreases over the course of a chirp, the probe intensity increases which eventually lead to congestion, called excursion. When an excursion occurs, pathChirp tries to calculate packet rate $\mathbf{R}_k$ through equation $\mathbf{R}_k = \mathbf{P}/\Delta_k$, where k denotes packet index and $\mathbf{R}_k$ denotes available bandwidth. However, not all excursions are indication of congestion. Transient excursions may occur due to variance in the underlying communication channels. pathChirp filters such transient excursions out by applying statistical analysis on the packet delays over a certain period of time, called a sliding window $\mathbf{W}$.

To analyze the performance of chirps in Wi-Fi environment, we constructed a simple testbed as depicted in Figure 3. The testbed is comprised of three components: smartphone client (measurement host), wireless AP and measurement server. The Measurement server is connected to the AP through 1 Gbps Ethernet. Samsung Galaxy S3 (Jelly Bean 4.1.2) is adopted as

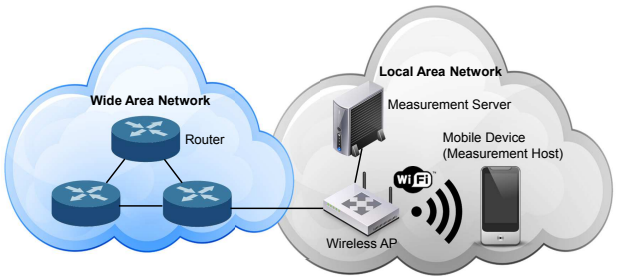


Fig. 3. Testbed Setup for Studying pathChirp

the measurement host, which is equipped with BCM4334 Wi-Fi communication module. The original pathChirp is implemented in C and runs on top of linux machine (measurement host and server). We ported the client program to Android smartphones using Android's Native Development Kit (NDK). The pathChirp server program is unchanged and resides on the measurement server.

B. Probe Packet Size

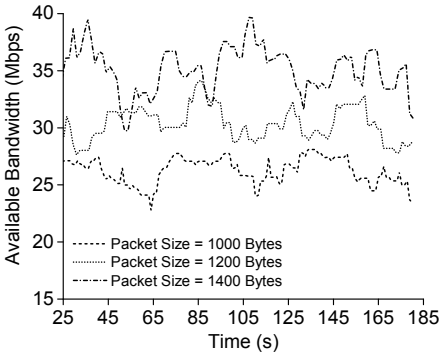
Probe packet size, which denotes the size of each packet per chirp, is regarded as one of the most important parameters that affect measurement result. According to the relation between $\mathbf{P}$ and $\mathbf{R}_k$, when we increase $\mathbf{P}$, $\mathbf{R}_k$ will increase accordingly (see Figure 4(a) and 4(b)).

The smaller size of packet causes underestimated measurement result in the majority of available bandwidth measurement tools. Let the average packet rate within time τ be $\mathbf{R}_{avg}$, let l denote the chirp index and let N_c denote the number of chirps generated within time τ. The number of packets in each chirp is noted as N_p. When we have smaller size of packet $\mathbf{P}$, according to Equation 1, to preserve the same packet rate, the overall probe rate $(N_c \times N_p)/\tau$ should be increased accordingly. Note that probe rate stands for the number of probe packets transmitted per second.

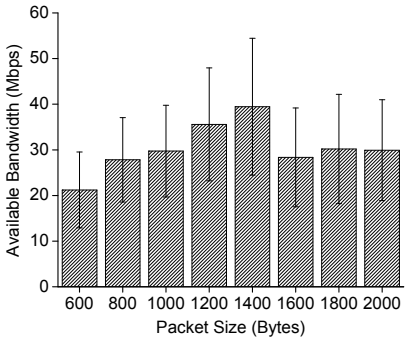
$$\mathbf{R}_{avg} = \frac{\sum_{l=1}^{N_c}\sum_{k=1}^{N_p}\mathbf{P}_{k,l}}{\tau} \quad (1)$$

In [14], it is shown that due to the computing power constraint, the existing tools always underestimate available bandwidth, and such underestimation is mainly caused by the increased probe rate. Low achievable probe rate limits the maximum achievable throughput for small packets, therefore, smaller packet size induces less accurate measurement result, especially for the devices which have limited computing power. Since we use smartphone as measurement host, intuitively, choosing larger packet size would overwhelm less the smartphone, therefore, we can obtain more accurate result.

Although larger packet size yields more accurate result, however, we cannot enlarge the packet size unlimitedly. The reason is if the packet size reaches the Maximum Transmission Unit (MTU), defined by IEEE 802.3 protocol, the protocol will split the packet into several fragments. Assume that we have a packet with size 1728 bytes and in which packet header occupies 20 bytes, then according to IEEE 802.3 standard defined



(a) Available Bandwidth Measurement Result with Different Probe Packet Size



(b) Available Bandwidth Statistics with Different Probe Packet Size

```
01  ...
02  Header checksum: 0x38bb [correct]
03  Source: 192.168.0.152
04  Destination: 192.168.0.153
05  [2 IPv4 Fragments (1708 bytes): #16(1480),
06  #17(228)]
07    [Frame: 16, payload: 0-1479 (1480 bytes)]
08    [Frame: 17, payload: 1480-1707 (228 bytes)]
09    [Fragment count: 2]
10    [Reassembled IPv4 length: 1708]
11  User Datagram Protocol, Src Port: 8365, Dst
12  Port: 53227
13  Data (1700 bytes)
14  ...
```

(c) Deep Packet Inspection through Wireshark

Fig. 4. Available Bandwidth Analysis with Respect to Probe Packet Size

(e.g., MTU = 1500 bytes), the packet will be fragmented into 1500 bytes and 228 bytes respectively. Since two fragments share the same packet header, the size of payload in each fragment is 1480 bytes and 228 bytes respectively. We validate our analysis result through wireshark application with 1728 sized packet, and the validated result is shown in Figure 4(c). Since in pathChirp, we only deal with the average packet rate, so the measured bandwidth will also be taken from the average packet rate of fragments, which causes inaccurate measurement result. The packet with size $\widetilde{\mathbf{P}}$ bytes, approximately has same packet rate as that with size $\lceil \widetilde{\mathbf{P}}/\mathbf{MTU} \rceil$ bytes. To preserve the measurement accuracy, we set the size of payload in all packets as 1480 bytes in following experiment.

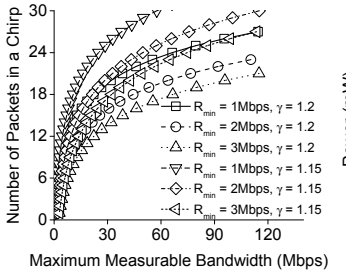
C. Spread Factor & Measurable Bandwidth

pathChirp has a number of tunable parameters, among which spread factor γ and measurable bandwidth range (defined by maximum measurable bandwidth $\mathbf{R}_{max}$ and minimum measurable bandwidth $\mathbf{R}_{min}$) are of particular importance to us because they jointly determine the rate of probe packet generations. Based on these three parameters, Inter-spacing time Δ is determined, which is exponentially decreasing across packets in the same chirp train.

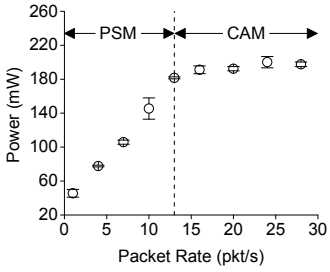
For instance, assuming we have N packets per chirp, then the first inter-spacing time Δ_2 will have $T\gamma^{N-2}$ ($\gamma > 1$ strictly), while the last inter-spacing time Δ_N will have value T. The following equation is used to determine all Δ within a chirp:

$$\mathbf{\Phi} = \sum_{i=2}^{N} \Delta_i = \sum_{i=2}^{N} T\gamma^{N-i} = T\sum_{i=2}^{N} \gamma^{N-i} \qquad (2)$$

Each chirp has identical cumulated inter-spacing time, i.e., $\mathbf{\Phi} = 1$ (second). Therefore N varies with γ, and hence varied probing packet generation rate. Since the packet rate directly relates to energy consumption and communication overhead, we want to quantify the relation among γ, $\mathbf{R}_{min}$, $\mathbf{R}_{max}$, and N. Accordingly, we have performed series of experiments based on the following setup: for differing the measurable bandwidth range, we linearly increase the value of $\mathbf{R}_{max}$ from 3 Mbps to 120 Mbps, and assign 1 Mbps, 2 Mbps and 3 Mbps to $\mathbf{R}_{min}$ respectively. Moreover, we choose 1.15 and 1.2 for γ and conduct six sets of experiments. In order to track the packet generation rate, we use a packet count program and compute metrics on generated packets per second. Each set of experiments is performed multiple times and the averaged value is computed and shown in Figure 5(a). On Figure 5(a), we can observe that with the same measurable bandwidth range (the line for the same $\mathbf{R}_{min}$), as value of γ increases, smaller number of packets are generated, while with the same γ value, smaller range of measurable bandwidth (which has larger $\mathbf{R}_{min}$) generates smaller number of packets.



(a) The relation between minimum measurable bandwidth $\mathbf{R}_{min}$, spread factor γ and number of packets generated per second



(b) The relation between energy expenditure and number of packets transmitted per second

Fig. 5. The relation between energy expenditure and pathChirp parameters

The Wi-Fi two operational modes' switching is determined by the packet transmitting rate. However, as we can see in Figure 5(b), even with a large value of γ and low value of $\mathbf{R}_{min}$, pathChirp switches the network device into CAM mode. Now, if we can control the packet generation rate within the mode switching threshold, we can save significant amount of energy. This is the key rationale behind BreezChirp. However, the measurement efficiency suffers significantly when we do this. Hence novel technique is needed to compensate for this loss of efficiency.

D. BreezChirp

BreezChirp constrains the chirp to operate under PSM mode with an adaptive sliding window technique, and compensate the loss of efficiency with application traffic adaptive full chirps. In doing so, we arrive at a new bandwidth measurement tool that is energy efficient, effective, and suitable for smartphones. Based on experiments, we found that 8

packets per second is the threshold value that triggers CAM to PSM switch. Conversely, 13 packets per second is the threshold value that triggers PSM to CAM switch. Therefore, if we can control the chirp packet generation rate per second to below this threshold, we can achieve energy efficiency. However, this in practice severely constrains the bandwidth probing range of a chirp. If there is a drastic change in the environment in terms of available bandwidth (e.g., due to mobility, interference, cross traffic, etc.), BreezChirp cannot efficiently shift its measurement range within reasonable time frame. Therefore, full chirp is needed to compensate for this loss of efficiency. Again as we observe in Figure 5(b), when the network device is in CAM mode, increasing the packet rate does not significantly increase energy expenditure. Hence BreezChirp schedules a full chirp measurement to coincide with application-level traffic activities, by detecting whether the network device is already in CAM mode or not. Thus the design of BreezChirp includes two modes: *limited* mode (with a narrow sliding window), efficient for measuring moderate bandwidth changes in the environment; and *normal mode* (full chirp), efficient in capturing drastic bandwidth changes.

As γ, the spread factor, moderates the packet generation rate. We want to be able to constrain its value to fit into the PSM threshold. The way pathChirp computes available bandwidth is as follows: $\mathbf{R}_k = \mathbf{P}/\Delta_k$ is used to determine the packet rate and when channel congestion occurs at a particular packet rate, we obtain the available bandwidth. We can rewrite the equation with respect to γ as,

$$\mathbf{R}_{k+1} = \frac{\mathbf{P}}{T\gamma^{N-(k+1)}} = \frac{\mathbf{P}}{T\gamma^{N-k}} \times \gamma = \mathbf{R}_k \times \gamma \qquad (3)$$

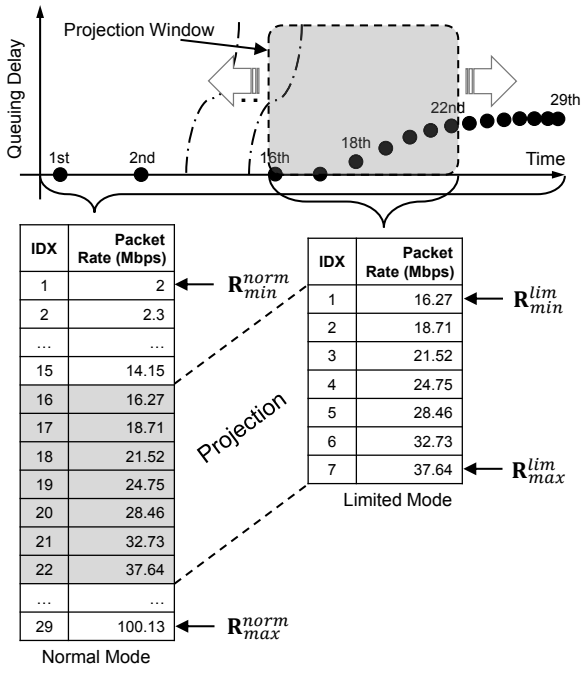
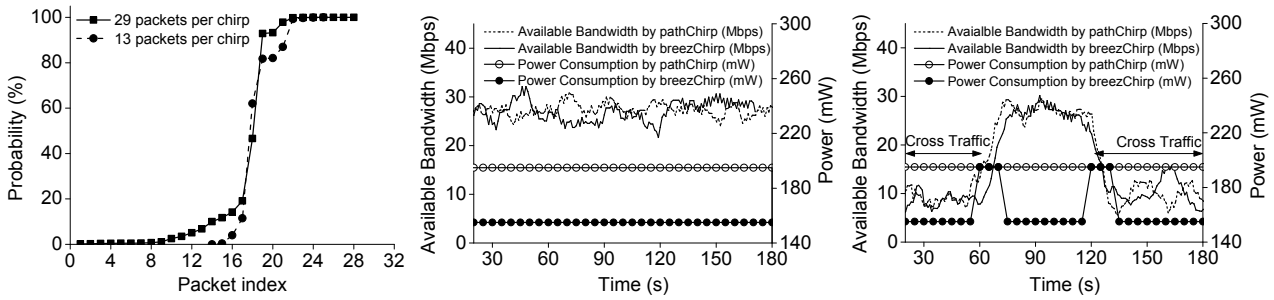


Fig. 6. Example of Adaptive Range Projection

In Equation 3, k is the index number of a packet within a chirp, and the next packet rate $\mathbf{R}_{k+1}$ is computed by multiplying γ to the current packet rate $\mathbf{R}_k$. As γ increases, the inter-packet gap of two consecutive packets increases accordingly, which in turn reduces the measurement granularity. Because of this inverse relation between γ and measurement granularity, we cannot increase γ without bound. This led us to choose the most appropriate value of γ with which we can preserve maximum measurement granularity while generating the least number of packets. Through numerous experiments, with measurement bandwidth range $0 \sim 100$ Mbps, we found that 29 is the least number of packets, and we denote the spread factor that yields 29 packets as γ_{opt}. However, 29 packets per second will put the network device into CAM mode which we do not want, we therefore need to also adjust the measurable bandwidth range to further reduce the packet number from 29 to 13. Since the measurable bandwidth range covered by 13 packets is the subset of that covered by 29 packets, we term this subset a *projection*. BreezChirp's *limited* mode utilizes the technique of *adaptive range projection*, which is illustrated in Figure 6. It operates like a sliding window that dynamically adjusts to a projection that encapsulates the available bandwidth. *Limited* mode uses the least amount of packets $\mathbf{N}_{lim}$ (13) to measure a narrow range of bandwidth ($\mathbf{R}_{min}^{lim}$ - $\mathbf{R}_{max}^{lim}$ Mbps). Note tat, for the sake of space limitation, we only show $\mathbf{N}_{lim}$ = 8 case in Figure 6, in which BreezChirp only uses 8 packets per in second in *limited* mode for the purpose of ensuring energy saving.

The pseudo code of the adaptive range projection algorithm is given in Algorithm 1. The algorithm is comprised of four parts: 1) Check whether the currently measured bandwidth is out of the newly adapted measurement range of *limited mode*. If it is out of the range, try to reset the range and return to *normal mode* (line 1-4). Otherwise, try to adapt the new measurement range based on the currently measured bandwidth (line 5-17); 2) Divide the $\mathbf{N}_{lim}$ into three partitions - $\mathbf{N}_{lim}^{up}$, $\mathbf{N}_{lim}^{mid}$ and $\mathbf{N}_{lim}^{low}$ (line 6-8); 3) Find the packet rate $\mathbf{R}_{appr}$ which is closely approximated to current measurement result $\mathbf{R}_{curr}$ and assign $\mathbf{N}_{lim}^{mid}$ as the index of $\mathbf{R}_{appr}$ (line 9-11); and 4) Find the new measurement range through calculating $\mathbf{R}_{min}^{lim}$ as well as $\mathbf{R}_{max}^{lim}$ by dividing spread factor $\mathbf{N}_{lim}^{up}$ times to $\mathbf{R}_{appr}$, or multiplying spread factor $\mathbf{N}_{lim}^{low}$ times to $\mathbf{R}_{appr}$ (line 12-17). If the new measurement result is larger than the previous result, the algorithm would move the projection window forward to preserve the increasing trend. On the contrary, the algorithm would move the projection window backward to preserve the decreasing trend (see Figure 6). In this way, *limited* mode adapts the measurement range slowly according to the newest measurement result, and is ideal for accommodating small to moderate changes in the environment with respect to available bandwidth.

However if the bandwidth change is drastic in the environment, adaptive range projection would not work well. We can draw similarity to how an observer tracks a moving target in the distance with a binocular. When the target is moving slowly, the observer can track the target through his binocular, which has precise focus but very limited scope. However, if the target all of a sudden vanishes from within the scope (e.g. a sudden dash). The observer will use his eyesight to locate the target's new location rather than trying to search with the binocular which is too slow to adapt. Similarly, we need a full chirp to find the available bandwidth when the bandwidth change is too drastic. BreezChirp's *normal* mode does exactly



(a) PDF of congestion with 29 and 13 packets in a chirp

(b) Measured Available Bandwidth and Energy Consumption Comparison without Cross Traffic

(c) Measured Available Bandwidth and Energy Consumption Comparison with Cross Traffic

Fig. 7. Comparison Result of Measured Available Bandwidth Accuracy and Energy Consumption from pathChirp and BreezChirp

Algorithm 1: Adaptive Range Projection Algorithm

input : stable measurement result $\mathbf{R}_{curr}$,
number of packets for limited mode $\mathbf{N}_{lim}$
output: minimum packet rate for limited mode $\mathbf{R}_{min}^{lim}$,
maximum packet rate for limited mode $\mathbf{R}_{max}^{lim}$

/* Check whether $\mathbf{R}_{curr}$ is out of range */
if $\mathbf{R}_{curr} < \mathbf{R}_{min}$ ***or*** $\mathbf{R}_{curr} > \mathbf{R}_{max}$ **then**
  $\mathbf{R}_{min}^{lim} \leftarrow \mathbf{R}_{min}^{norm}$;
  $\mathbf{R}_{max}^{lim} \leftarrow \mathbf{R}_{max}^{norm}$;
  go to normal mode;
else
  /* Partition $\mathbf{N}_{lim}$ into three parts */
  $\mathbf{N}_{lim}^{mid} \leftarrow 1$;
  $\mathbf{N}_{lim}^{up} \leftarrow \lceil \mathbf{N}_{lim}/2 \rceil - 1$;
  $\mathbf{N}_{lim}^{low} \leftarrow \mathbf{N}_{lim} - \mathbf{N}_{lim}^{up} - \mathbf{N}_{lim}^{mid}$;
  /* Find the appr. measurement range */
  $\mathbf{R}_{appr} \leftarrow \mathbf{R}_{min}^{lim}$;
  while $\mathbf{R}_{appr} < \mathbf{R}_{curr} - \epsilon$ **do**
    $\mathbf{R}_{appr} \leftarrow \mathbf{R}_{appr} \times \gamma$;
  /* Seek the new measurement range */
  $\mathbf{R}_{max}^{lim} \leftarrow \mathbf{R}_{appr}$;
  $\mathbf{R}_{min}^{lim} \leftarrow \mathbf{R}_{appr}$;
  for $i = 1, 2, 3, ..., \mathbf{N}_{lim}^{up}$ **do**
    $\mathbf{R}_{max}^{lim} \leftarrow \mathbf{R}_{max}^{lim} \times \gamma$;
  for $i = 1, 2, 3, ..., \mathbf{N}_{lim}^{low}$ **do**
    $\mathbf{R}_{min}^{lim} \leftarrow \mathbf{R}_{min}^{lim} / \gamma$;

that. Energy efficiency is achieved by: 1) Use $\mathbf{N}_{norm}$ (29) to measure the available bandwidth in full measurable bandwidth range ($\mathbf{R}_{min}^{norm}$ - $\mathbf{R}_{max}^{norm}$ Mbps). As we discussed before, 29 packets appears to be the least number of packet per second needed to obtain a good measurement in Wi-Fi environment; 2) Whenever possible, we schedule full chirp when there is active application traffic (i.e., the network device is already in CAM mode).

The switching between *limited* mode and *normal* mode is implemented as follows: at initial stage, we have no clue as to where the bandwidth range for *limited* mode should reside, and therefore we obtain a stable measurement from *normal* mode. We obtain a stable measurement result through observing the variance of measurement results in a period $\mathbf{W}$, and if the variance is smaller than the predefined threshold, then we regard the measurement result as stable result. Once we have a stable result, *limited* mode is activated by setting the projection such that the mid-bracket projection covers the observed bandwidth measurement. In *limited* mode, we continuously update the measurement range until a new measurement result is out of all of the measurement brackets, then we schedule a *normal* mode. In this way, BreezChirp is able to provide efficient Wi-Fi bandwidth measurement while conserving energy.

V. Evaluation

In this section, we evaluate the performance of BreezChirp and BattMan that implements the two energy saving schemes - application packing and alignment. Current implementation of BattMan is targeted for Android platform and it uses network monitor implemented as a kernel module. We use the same experiment setup as shown in Section IV, and adopt an Android based power measurement tool - PowerTutor [2] to measure the power consumption by smartphone in following experiments. The real-world experiment setup is shown in Figure 8.

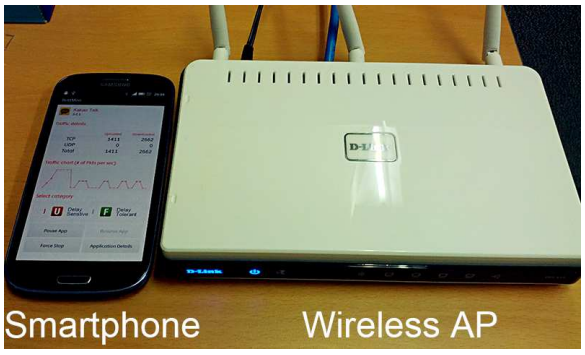


Fig. 8. Real World Experiment Setup with BattMan and BreezChirp

A. BreezChirp Evaluation

To evaluate the performance of BreezChirp, we ported pathChirp and implemented BreezChirp on an Android phone.

First we examine the impact of reducing packet numbers on measurement accuracy. Figure 7(a) shows the probability of congestion being detected by a chirp on a given packet index in the chirp. The available bandwidth of the measured channel is indicated by the sharp rise in congestion probability. When the available bandwidth is within the bandwidth measurement range, 13 packets per second is sufficient.

Two experiments are performed for the purpose of evaluating the bandwidth measurement accuracy and energy efficiency of BreezChirp. The first experiment is conducted in wireless environment in which channel condition is relatively stable. Figure 7(b) shows the measured available bandwidth and energy consumption with pathChirp and BreezChirp for the first experiment. BreezChirp performs as well as pathChirp but at a fraction of the energy consumption.

The second experiment is conducted in wireless environment with synthesized cross traffic by using FTP file transmission workload. The traffic is generated from another terminal which is connected to the same wireless AP as the terminal running BreezChirp. The traffic generation is commenced at the beginning of experiment, suspended on 1 min mark, and resumed on 2 min mark. We therefore created two environmental transitions. The result is shown in Figure 7(c), and as we can see from the Figure, BreezChirp adapts to the changes very well. We observe a short lag when the environment transition occurs because a switch from *limited* mode to *normal* mode have occurred in BreezChirp. Overall, BreezChirp conserved around 20.6% power consumption compared to pathChirp in this experiment, and this rate increases with prolonged usage of BreezChirp as it achieves higher energy saving rate when the environment condition is stable.

B. BattMan Evaluation

We now utilize the bandwidth metric provided by BreezChirp in BattMan that implements application packing and alignment. In the first experiment, we performed packing on delay tolerant applications. Three different applications - Dropbox, Download Manager and AndFTP are used in this experiment, where each of which has 300 Kbps, 200 Kbps and 500 Kbps throughput and 8 second, 18 second, and 58 second task (transmission) arrival time respectively. BreezChirp clocks the average available bandwidth during the first experiment to be around 450 Kbps. In uncontrolled case (without BattMan), power consumption spikes up (CAM mode) whenever transmission occurs, while in controlled case (with BattMan), application transmissions are synchronized. Figure 9 shows the resulting power consumption measurements. As we can observe, the first application is postponed around 10s until the second application arrives. Since the total amount of expected throughput of Dropbox and Download Manager exceeds available bandwidth, the data transmission of Dropbox and Download Manager is permitted. Overall, we obtained around 42.4% energy reduction.

In the second experiment, we have a mix of delay sensitive and delay tolerant applications. We adopt Facebook as a delay sensitive application, while a background Download Manager as a delay tolerant application.

In the uncontrolled scenario, regardless of the behavior of Facebook, Download Manager always tries to download the data, while in the controlled scenario, Download Manager is delayed intermittently in order to sync with Facebook's transmission. As we can observe from the overall power consumption graph shown in Figure 10, in uncontrolled scenario, the overall throughput has a consecutive shape and lasts 60 seconds. While three delay phases are detected when BattMan is used. An additional 20 seconds are spent in controlled case in order to download the same amount of data as in uncontrolled case for preserving fairness. We achieve around 23.3% energy reduction in alignment scenario.

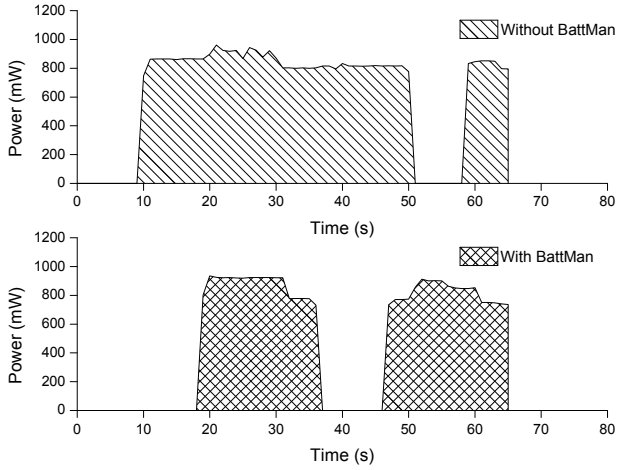


Fig. 9. Energy Expenditure Comparison on Application Packing

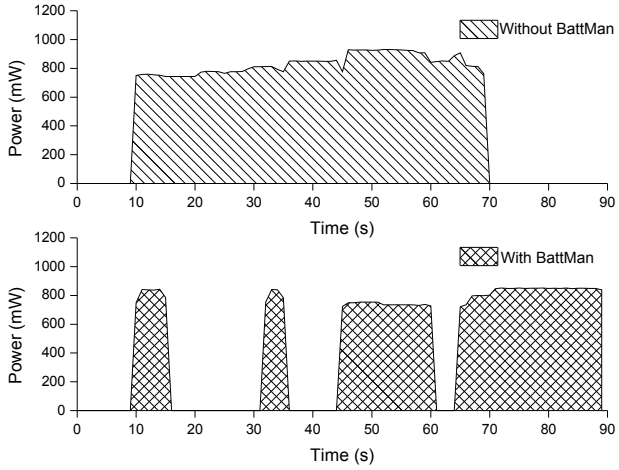


Fig. 10. Energy Expenditure Comparison on Application Alignment

VI. Conclusion

In this paper, we presented an application-aware approach to Wi-Fi energy management for smartphones. We started by investigation energy consumption patterns in Wi-Fi environment and found that PSM prolongation and CAM bundling are the key energy saving principles. Accordingly we have designed BattMan an energy management mechanism which implements application packing and alignment schemes. We further found that an energy efficient and accurate Wi-Fi bandwidth measurement tool is fundamental to the efficiency of BattMan and have thus developed BreezChirp as a companion solution. Through on device implementation and real world field experiments, we have evaluated the performance of both BreezChirp and BattMan and have found that indeed they offer a comprehensive solution that can significantly reduce energy consumption due to Wi-Fi communication on smartphones. As future work, we plan to extend our work to include 4G radio communication.

REFERENCES

[1] 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.

[2] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Proceedings of the 8th IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, 2010, pp. 105–114.

[3] J. Manweiler and R. Roy Choudhury, "Avoiding the rush hours: Wifi energy management via traffic isolation."

[4] X. Chen, S. Jin, and D. Qiao, "M-psm: Mobility-aware power save mode for ieee 802.11 wlans," in *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*, june 2011, pp. 77 –86.

[5] M. Li, M. Claypool, and R. Kinicki, "Wbest: A bandwidth estimation tool for ieee 802.11 wireless networks," in *Local Computer Networks, 2008. LCN 2008. 33rd IEEE Conference on*, oct. 2008, pp. 374 – 381.

[6] M. Bredel and M. Fidler, "A measurement study of bandwidth estimation in ieee 802.11 g wireless lans using the dcf," in *NETWORKING 2008 Ad Hoc and Sensor Networks, Wireless Networks, Next Generation Internet*, 2008, pp. 314–325.

[7] V. J. Ribeiro, R. H. Riedi, R. G. Baraniuk, J. Navratil, and L. Cottrell, "pathchirp: Efcient available bandwidth estimation for network paths," in *Passive and Active Measurement (PAM)*, 2003.

[8] F. R. Dogar, P. Steenkiste, and K. Papagiannaki, "Catnap: Exploiting high bandwidth wireless interfaces to save energy for mobile devices," in *MobiSys '10*, 2010, pp. 107–122.

[9] C. Dovrolis, P. Ramanathan, and D. Moore, "What do packet dispersion techniques measure?" in *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 2, 2001, pp. 905 – 914.

[10] M. Jain and C. Dovrolis, "End-to-end available bandwidth: measurement methodology, dynamics, and relation with tcp throughput," *IEEE/ACM Transactions on Networking (TON)*, vol. 11, no. 4, pp. 537 – 549, aug 2003.

[11] B. Melander, M. Bjorkman, and P. Gunningberg, "A new end-to-end probing and analysis method for estimating bandwidth bottlenecks," in *Global Telecommunications Conference*, 2000, pp. 415 – 420.

[12] A. Johnsson, B. Melander, and M. Björkman, "Diettopp: A first implementation and evaluation of a simplified bandwidth measurement method," *Swedish National Computer Networking Workshop*, 2004.

[13] J. Li, J. Xiao, J. W.-K. Hong, and R. Boutaba, "Application-centric wi-fi energy management on smart phone," in *Asia-Pacific Network Operations and Management Symposium*, sept. 2012, pp. 1 – 8.

[14] O. Goga and R. Teixeira, "Speed measurements of residential internet access," in *Passive and Active Measurement (PAM)*, 2012, pp. 168 – 178.