# Characterizing Smartwatch Usage In The Wild

Xing Liu[1] Tianyu Chen[1] Feng Qian[1]

Zhixiu Guo[23] Felix Xiaozhu Lin[4] Xiaofeng Wang[1] Kai Chen[25]

[1]Indiana University Bloomington
[2]SKLOIS, Institute of Information Engineering, Chinese Academy of Sciences
[3]Beijing Jiaotong University [4]Purdue ECE
[5]School of Cyber Security, University of Chinese Academy of Sciences

## ABSTRACT

Smartwatch has become one of the most popular wearable computers on the market. We conduct an IRB-approved measurement study involving 27 Android smartwatch users. Using a 106-day dataset collected from our participants, we perform in-depth characterization of three key aspects of smartwatch usage "in the wild": usage patterns, energy consumption, and network traffic. Based on our findings, we identify key aspects of the smartwatch ecosystem that can be further improved, propose recommendations, and point out future research directions.

## CCS Concepts

**•Human-centered computing → Ubiquitous and mobile devices;**

## 1. INTRODUCTION

Wearable devices are weaving computing into our daily lives. Among diverse wearables, smartwatches are probably one of the most popular gadgets. According to a recent CNET article [10], smartwatch sales are expected to jump from 30 million in 2015 to 50 million in 2016 and then to 67 million in 2017. Smartwatches offer great convenience to end users through a wide range of features such as receiving push notifications, issuing voice control, monitoring fitness, and interacting with 3rd-party apps. Despite their popularity, smartwatches are still relatively new to the commercial mobile device family, and the research community lacks a thorough understanding of the smartwatch ecosystem.

This paper bridges the above gap by conducting an IRB-approved crowd-sourced measurement study of smartwatches involving 27 users. We first demonstrate it is feasible to build a self-contained and comprehensive measurement data collector on today's off-the-shelf Android smartwatches. The collector transparently collects a wide range of usage data, network traffic, and system events in realistic usage scenarios, with very low runtime and energy overhead incurred (§3). We then provide each of the 27 users with a state-of-the-art smartwatch instrumented with the data collector. Using a 106-day dataset collected from our participants, we conduct an in-depth characterization of three key aspects of smartwatch usage "in the wild": usage patterns, energy consumption, and network traffic characteristics. This is to our knowledge the most comprehensive and in-depth crowd-sourced study of smartwatches. Our key measurement results consist of the following.

• We characterize the smartwatch usage patterns. An Android watch can stay in one of the four states with diverse power characteristics: fully awake, dozing (with dimmed watch face display and restricted system activity), sleeping (screen further turned off), and charging. We find that smartwatch's wake-up period accounts for only 2% of the overall usage period among the four states. The wake-up sessions are not only short, but also frequent (72 times per day on average). We then analyze their triggering factors, which help us identify key usage scenarios of a watch: short "flick and look" sessions (the most common interaction type that the OS needs to be optimized for), push notifications, longer interaction sessions, and unintended wake-up (§4.1).

• A key usage scenario of smartwatches is to receive push notifications. In our dataset, more than 200 apps send push notifications to the watch through either the OS-provided Android Wear service or custom data channels between phone-side and watch-side apps. Push notifications are dominated by instant messaging and emails. Despite a potential lack of long-term predictability, push notifications' arrival exhibits a strong bursty pattern, with a median inter-arrival time of only 49 seconds. Also there is room for the OS to improve push notification delivery, by strategically determining whether to push and how to push (§4.2).

• We also investigate how smartwatch applications (apps) behave. We find smartwatch app execution is dominated by short but frequent background services, whose total duration is more than 50 times longer than that of full-screen activities, which users seldom launch due to a watch's small form factor making full-fledged interaction challenging. The services, together with the OS infrastructures (*e.g.,* the push notification and card management subsystems) should therefore be the optimization focus for OS and app developers (§4.3).

• We derive comprehensive and accurate power models for two popular Android smartwatches (§5.1). We then apply the power models to the user study data to quantify the energy consumption of smartwatches in the wild. We made several interesting observations. More than half of the smartwatch energy is consumed by the dozing state (56%) due to its long duration. Meanwhile, the awake state also plays an important role in energy consumption (27%) despite its very short usage duration (2%). Due to the



*MobiSys'17, June 19-23, 2017, Niagara Falls, NY, USA*



DOI: http://dx.doi.org/10.1145/3081333.3081351

| § | Topic | Key Results |
|---|---|---|
| §4.1 | Device states | Dozing dominates the usage (50.6%); wake-up accounts for 2% of usage period; wake-up sessions are short but frequent (72 per day on average) with various root causes such as "flick and look" and push notifications. |
| §4.2 | Push notifications | Push notifications are used by 200+ apps, dominated by IM/emails, and exhibiting bursty arrival patterns. |
| §4.3 | Smartwatch apps | A wide spectrum of watch apps are observed; their execution is dominated by short background services. |
| §5.1 | Power models | Accurate and comprehensive power models for two popular watches. The models have error rates $<$6%. |
| §5.2 | Energy utilization | Dozing and wake-up account for 56% and 27% of overall energy respectively. At component level, CPU (29%) and display (30%) dominate the energy consumption. Network consumes only 3.4% of the energy. |
| §5.3 | Energy optimization | "What-if" analysis over real data for improving energy efficiency; impact of state machine on battery drain. |
| §6 | Network traffic | Watches are paired with phones during 84% of the daytime; most flows are small, short, slow, and bursty. |

Table 1: Key results and findings of our study.

big power consumption gap between awake and dozing/sleeping, a small increase of the wake-up duration will be "amplified" from the battery draining perspective, thus shortening the standby time of the watch. The top-2 energy-hungry components on smartwatches are the same as those on smartphones: CPU and display. Despite smartwatches' small screen sizes, display still contributes to 30.2% of the overall energy. CPU accounts for 29.3% of the overall smartwatch energy. Network interfaces, however, consume a very small fraction of energy on smartwatches without a cellular interface (§5.2).

• We investigate how to make smartwatches more energy-efficient using combined approaches of trace-driven "what-if" analysis and controlled in-lab experiments. Our results suggest that the energy efficiency can be further improved through a wide range of optimization strategies such as optimizing the display, bundling delay-tolerant push notifications, and dynamically configuring CPU's online cores and frequencies. Also the watch's state machine (Figure 1) and its associated timers affect the energy consumption and need to be strategically determined, as demonstrated by our "what-if" analysis (§5.3).

• We conduct a first measurement study of smartwatch network traffic. We find that watches are paired with phones during 84% of the day time, making Bluetooth traffic account for 91% of the overall smartwatch traffic. Smartwatch traffic is dominated by downlink traffic (from phone/Internet to the watch). The vast majority of network flows are small, short, slow, and bursty, with the flow size and rate being very well correlated. Our characterizations provide hints and knowledge for improving the network protocol stack for future wearable devices (§6).

Overall, our findings suggest that in many aspects (user behaviors, app execution patterns, energy consumption, network traffic, *etc.*), smartwatches exhibit characteristics that are different from those of smartphones, which have been extensively studied in the literature. Our quantitative measurement results complement the qualitative smartwatch app design guidelines published by major vendors [4, 11, 8]. Our key findings are summarized in Table 1. To summarize, our contributions consist of the following.

• We develop a self-contained and lightweight measurement data collection system for Android Wear smartwatches, and conduct a real deployment on 27 users.

• We derive accurate and comprehensive power models for two commodity Android Wear watches.

• We perform systematic measurements of smartwatches' usage patterns, energy consumption profiles, and network traffic characteristics using a 106-day dataset collected from 27 users. Based on our findings, we identify key aspects in the smartwatch ecosystems that can be further improved, and propose recommendations and future research directions.

## 2. RELATED WORK

**Understanding Mobile Devices "in the Wild".** Researchers have carried out numerous crowd-sourced measurements of smartphones [28, 20, 52, 7, 46, 49, 39, 14]. While they partially motivate our user trial, much fewer efforts have been made for wearable devices. Among them, Gouveia *et al.* studied how users engage with activity trackers [22]. Lazar *et al.* interviewed 17 participants to study the incentives of using and abandoning smart devices [29]. Lyons studied usage practices of traditional dumb watches by conducting a user survey [34]. Min *et al.* characterized smartwatch battery use based on online survey and a user study involving 17 users [37]. They studied users' satisfaction toward smartwatch life, charging behaviors, and interaction patterns. In a recent extended abstract [44], Poyraz *et al.* also described their smartwatch user study involving 32 users for 70 days. Using the collected data, they analyzed the watches' power consumption and characterized user activities. While the detailed data collection and measurement methodologies were not fully documented, many of their findings such as the active state's high energy contribution compared to its short usage duration are qualitatively similar to ours. Compared to [37] and [44], we instead investigate a much wider spectrum of characteristics such as push notification, app usage, and network traffic by leveraging a much richer set of data items collected in Table 2. We also make our power models publicly available.

**Wearable System, Energy, and Applications.** LiKamWa *et al.* characterized the energy consumption of Google Glass [30]. Huang *et al.* proposed a fast storage system for wearables based on battery-backed RAM and offloading [25]. Santagati *et al.* designed an ultrasonic networking framework for wearable medical devices [50]. Miao *et al.* investigated the implications of smartwatches' circular display on resource usage [36]. Ham *et al.* proposed a novel display energy conservation scheme for wearables [24]. There also exists a large body of work from the mobile computing, sensing, and HCI community. These studies focus on novel wearable applications [51, 40, 35], wearable user interface design [15, 43], and wearable security [53, 33]. In contrast, our measurement focuses on characterizing wearable user behavior, application usage, energy consumption, and network traffic – all in the wild.

**Characterizing Android Wear OS.** Recently, Liu *et al.* analyzed the execution of Android Wear OS and presented a series of inefficiencies and OS implications [32, 31]. Compared to its in-lab, controlled experiments, this paper contributes the understanding of real-world wearable usage as well as the entailed design implications – backed by more solid evidence. Furthermore, our study characterizes key system aspects that were missing in the prior work, most notably power modeling and network behaviors.

| Collected Data Item | Method* | Source |
|---|---|---|
| Wi-Fi packet trace | E | tcpdump |
| Bluetooth packet trace | E | BT Snoop Logger |
| User input events | E | /dev/input/ |
| Voice control input event | E | Android Wear log |
| Device wakeup/doze/sleep | E | Android Wear log |
| Device charging | E | Android Wear API |
| Card post | E | Android Wear API |
| App activity/service state | E | atrace |
| Installed apps list | P (1 hour) | /data/app/ |
| Screenshot | P (30 sec) | screencap |
| CPU utilization | P (1 sec) | /proc/stat |
| Screen brightness level | P (5 sec) | Android Wear API |

*E = event triggered callback; P (interval) = periodical polling

Table 2: Types of data collected in the user study.

# 3. THE SMARTWATCH USER TRIAL

We launched an IRB-approved user study at Indiana University Bloomington. We made the study open to students, faculty, and staff members on our campus, and recruited 27 participants from more than 200 applicants. In our selection process, we tried to maximize the participants' diversity in terms of the occupation (student vs. non-student), gender, age, *etc.* An eligible participant must have an Android smartphone that can be paired with the watch. The selected voluntary participants consisted of 18 students (6 Ph.D. and 12 master students), 4 IU faculty members, and 5 staff members. 7 out of the 27 participants were female. Each user was provided with an LG Urbane watch, a high-end smartwatch as of early 2016 (cost ∼$250 USD). It is equipped with a quad-core Cortex A7 processor, 4GB storage, 512MB memory, Wi-Fi, Bluetooth, and various sensors. The watch runs Android Wear OS 1.3, which is based on Android 5.1.1. Android Wear is a version of Android OS tailored to small-screen wearable devices. Compared to an Android smartphone app, an Android Wear app has a different UI, but its developers still largely follow Android's programming paradigm. The wearable apps are written in Java and run on top of the managed Android Runtime [32]. Note that although our data is collected from Android Wear 1.3, when presenting the results, we do consider relevant new features to be added to Android Wear 2.0, the next version Android Wear (in its development preview stage at the time when this paper was written).

We face two challenges in deploying the study. The first challenge is, despite being very familiar with smartphones and conventional watches, many users had no or limited experience of using smartwatches. Ideally we do not want to collect data from a user who is still learning to use her smartwatch. We took two measures to address this. First, all users were required to attend an orientation session where we walked through all features of the watch and demonstrated popular smartwatch apps. Second, we started the actual data collection for this study at least two months *after* giving the watches to the users. This gives more than enough time for users to learn the watch usage and to develop their usage habits.

The second challenge is to build for the watches a data collector that is reliable, lightweight, and energy-efficient. Despite several mobile data collection systems having been developed in the literature [28, 20, 52, 46, 14], none of them was designed for smartwatches and was able to collect smartwatch-specific data such as card posts and push notifications. We therefore developed our own, which collects a wide range of data listed in Table 2 in the background. The data collector was written in Java and native C/C++ with about 6,500 LoC. Note the collector only runs at the watch side (the watch is rooted) and does not require any change on users' smartphones (which are usually not rooted). The collected data is automatically uploaded to our secure server over Wi-Fi at night when the watch is being charged. The data collection and upload processes are completely transparent to users. As listed in Table 2, most data items are collected using event-triggered callbacks, including network traffic (both packet header and payload), user input, card post[1], device status, application states, *etc.* In addition, the data collector performs periodic polling for a limited amount of other information, with the polling intervals being carefully chosen to balance between the runtime overhead and data collection frequency.

The data collector incurs very low runtime and energy overhead. We have measured that in common usage scenarios, its CPU and energy overhead (measured using Monsoon power monitor [5]) are less than 3%. The low overhead can be explained by several reasons. Although the collector captures many types of data, most types have low data rate and low arrival frequency. In particular, as to be measured in §6, the data rate of the network traffic is very low. We also strategically perform various system-level optimizations and polling interval tuning to reduce the collector's resource and computational footprint. It is also worth mentioning that compared to regular Android, the Android Wear OS enforces more aggressive sleeping policy for battery saving. During most of the time, the watch stays in "dozing" or "sleeping" mode where network and CPU-intensive services are suspended (see §4 for details). Therefore, our data collector is largely paused by the OS when the watch is dozing/sleeping so no periodical data is collected during that time. This brings little impact on the completeness of our data, because most data items collected periodically in Table 2 largely remain unchanged when the watch is dozing/sleeping. Meanwhile, the aggressive sleeping policy ensures the data collector incurs almost zero overhead during most of the time. Also the CPU periodically wakes up during the dozing/sleeping states and its utilization will be captured by the collector during that time.

We launched incremental deployment of the user study in spring 2016. Due to the incremental nature of our deployment, different users had different learning periods (at least two months) before we started the actual data collection. In this paper we analyze a 106-day dataset collected from 8/17/2016 to 11/30/2016. About 37 GB of data was collected during this period.

# 4. SMARTWATCH USAGE PATTERNS

In this section, we leverage our dataset to study how real users interact with the watch. We first provide some background of different states of an Android smartwatch. As shown in Figure 1, a watch can be in one of four states: dozing, sleeping, awake, and charging. During most of the time when a user wears a watch, the watch stays in the dozing state where user applications are suspended [6]. The user only sees a simplified watch face (*e.g.,* without the *second* hand) with low brightness, thus incurring very low power consumption (§5). Periodically, the watch exits dozing for a very short period of time (called "maintenance window") to let apps complete their deferred activities. The transition from awake to dozing is usually through an inactivity timer of a fixed duration (measured to be 5 seconds by default). The transition from dozing to awake is triggered by either user interaction or other external events (*e.g.,* the reception of a push notification). Also, if the watch is not worn (*e.g.,* left on a desk), it will enter from dozing to sleeping after a long timeout (measured to be about 35 minutes). In the sleeping mode, the display is further turned off. Note the watch does not sleep when it is worn unless the battery level is

[1]A *card* is a UI element in Android Wear. It shows a piece of information (*e.g.,* an incoming text message) to the user.

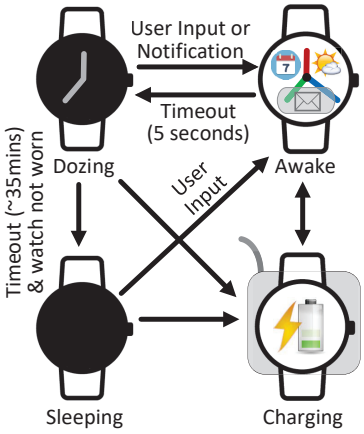


Figure 1: Device states for an Android smartwatch.

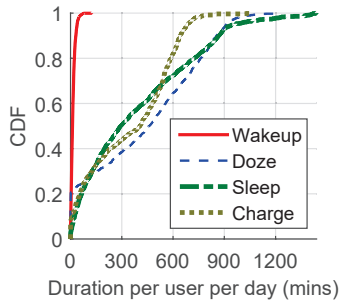


Figure 2: Distributions of the total daily duration of the four states that a watch stays at.

critically low. The transition to (from) the charging state is done by attaching the watch to (detaching it from) the charging dock.

## 4.1 Watch States and Basic Usage Patterns

We begin with understanding how long a watch stays at each state depicted in Figure 1. We calculate the daily awake/dozing/sleeping/charging duration across 24 hours starting from every midnight. As shown in Figure 2, the 25-th, 50-th, and 75-th percentiles of daily wake-up duration are 5.1, 11.6, and 19.6 minutes, respectively. In sharp contrast, the daily dozing duration tends to be much longer, with the 25-th, 50-th, and 75-th percentiles being 89, 460, and 711 minutes, respectively. The daily sleeping and charging time are also long. Table 3 shows the overall time breakdown for the four states. For half of the time, the watch is dozing while the wake-up time only accounts for 2% of the overall usage period. Despite this, the awake state is still very important due to two reasons. First, it is the key state where the user is actively interacting with the watch so providing good user experience is critical. Second, as shown in Table 3, the awake state contributes 27.2% of the overall energy consumption (to be characterized in §5), making its energy optimization also important.

**Characteristics of Wake-up Sessions.** Figure 3 plots the distribution of wake-up session duration across all users. A *wake-up session* starts when the watch wakes up, and ends when it dozes. As shown, the vast majority of wake-up sessions are short. The 5-second duration corresponds to the default inactivity timer for dimming the screen. We find that the wake-up sessions are not only short, but also frequent: the 25-th, 50-th, and 75-th percentiles of the number of wake-up sessions per user per

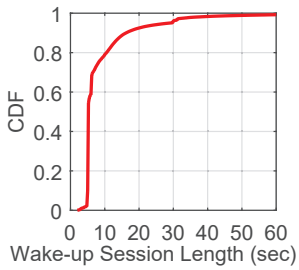


Figure 3: Wake-up session length distribution.

| State | Awake | Dozing | Sleeping | Charging |
|---|---|---|---|---|
| Duration | 2.0% | 50.6% | 24.7% | 22.8% |
| Energy | 27.2% | 56.0% | 16.8% | N/A |

Table 3: Overall usage duration and energy breakdown. The energy breakdown will be discussed in §5.

day are 34, 72, and 120, respectively. As to be studied shortly, wake-ups are triggered by different reasons and not all of them are noticed by the user. Note that prior measurements indicate smartphone interaction sessions are much longer: tens of seconds reported by [20] and even longer as measured recently by [48]. Such a striking difference between watches and phones can be explained not only by smartwatches' wearable nature enabling easy and ubiquitous access for users, but also by their tight coupling with smartphones that may frequently push notifications (chat, email, *etc.*) to watches. A recent study involving 17 smartwatch users also found that people interact with their watches very frequently but for a short time [37]. Despite their qualitatively similar findings, their reported numbers are a bit different from ours: in [37], users interact for an average of 20.6 minutes per day and the average session lasts for 13.0 seconds. Several factors such as users, watch type, and timeout settings may contribute to such quantitative differences.

**Wake-up Root Cause Analysis.** We next investigate which factors trigger a watch's wake-up by correlating each wake-up event $w$ with an external event $e$, which can be either a user input event (*e.g.,* gesture, screen touch, side-button press) or notification event. Doing so can help identify the watch's key usage scenarios. Let $T(x)$ be the timestamp when event $x$ occurs. If $T(w) - \delta \leq T(e) \leq T(w)$, then we assume $w$ is triggered by $e$. Based on controlled experiments, we set the detection window size $\delta$ to 1000ms while changing it to 500ms and 750ms yields qualitatively similar results. As shown in Table 4, 26.3% of wake-up sessions are triggered at the sleeping state when the watch is not worn. Such wake-ups are likely not seen by the user. For the remaining 73.7% of the sessions triggered at the dozing state (recall that at the dozing state, the watch is worn except during the 35-minute timeout), we classify all wake-up sessions into three categories. Category (a) consists of sessions that are triggered by user input and have the same session length as the timeout period (we use 6s instead of 5s to tolerate the duration of the user input itself). In other words, the user takes no further action after waking up the watch. Accounting for 43.5% of all sessions, this category corresponds to the watch's major usage scenario. We find such short sessions are mostly triggered by the "flick and look" action: 80.3% of sessions within Category (a) (35% of all sessions) are triggered by wrist gesture (as opposed to other user input types): when the user turns the watch toward herself to look at the watch face, the watch will wake up. It is likely that these short sessions are for a simple usage scenario: users take a quick glance at the watch to see the time, to

| Type | % | Median Len (sec) |
|---|---|---|
| Triggered when sleeping | 26.3% | 5.0 |
| Triggered when dozing | 73.7% | |
| .... (a) by user interaction, duration $< 6$s | .... 43.5% | 5.0 |
| ..... by wrist gesture | ..... 35.0% | 5.0 |
| ..... by other user input types | ..... 8.5% | 5.0 |
| .... (b) by user interaction, duration $\geq 6$s | .... 19.7% | 11.9 |
| .... (c) by notification | .... 10.5% | 6.1 |

Table 4: Breakdown of wake-up sessions.

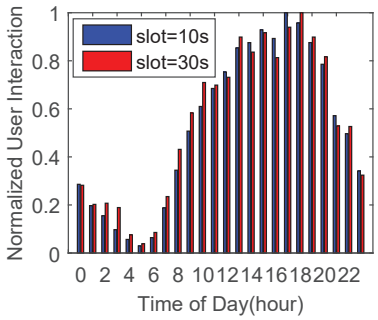


Figure 4: User interaction level during different hours.

| Category | % |
|---|---|
| Instant messenging | 43.3% |
| Email | 25.6% |
| Google Search box | 15.0% |
| Social media | 2.5% |
| Text message | 3.9% |
| Other | 9.8% |

Table 5: Application breakdown of push notifications.

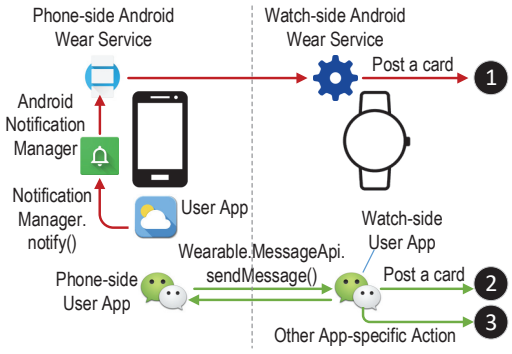


Figure 5: Communication between phone and watch.

check for notifications that she might have missed, or to see other information (*e.g.,* weather, temperature, or fitness records) that is directly displayed on the watch face (Figure 14). Also, although difficult to quantify, based on users' feedback, some of such "flick and look" sessions are unintended *i.e.,* due to the inaccuracy of the gesture sensing algorithm.

Category (b) in Table 4 consists of user-triggered sessions that are longer than the timeout period. Typical examples include browsing cards (recall a card is an Android Wear UI element), voice control, and interacting with smartwatch apps, which will be characterized in §4.3. Category (c) are those triggered by notifications. Upon the reception of a notification, the watch wakes up and pops up a card (Figure 9 left). We analyze push notifications in §4.2.

**Diurnal Usage Pattern.** We also investigate when our participants use the watch. The methodology is as follows: for each user, we discretize the timeline using fixed-length slots (*e.g.,* 10 seconds), and mark a slot if at least one user input event (touch, gesture, or voice) occurs within that slot. Figure 4 plots the user interaction level in terms of the normalized number of marked slots during different hours, across all users. As shown in Figure 4, despite the apparent diurnal pattern, there still exist some differences from smartphones' diurnal usage pattern. During daytime (9am to 9pm), the smartwatch usage is more uniformly distributed compared to that of smartphones [54, 20]. This is again explained by smartwatches' wearable and ubiquitous nature, as well as their function as an "on-wrist extension" of the phone.

## 4.2 Push Notification

This section characterizes push notification, an important application on smartwatches. We begin with explaining how an Android watch communicates with its paired phone. There are two ways as explained in Figure 5. For a smartphone app that does not have its watch version, when the app posts a notification[2], the phone-side Android Wear service will automatically and transparently push this notification to the watch-side service. Since at the time when this paper was written, most smartphone apps do not have their watch versions, this (the red arrows in Figure 5) is a common way for allowing *one-way* communication from the phone to its paired watch. If a phone app has its watch version, they can leverage Android Wear APIs [9] to perform *two-way* communication by exchanging arbitrary data, as illustrated by the green arrows. When the watch-side Android Wear service receives a notification, it will post a card and wake up the watch (❶ in Figure 5). The watch-side app may either post a card (❷) or take other app-specific actions (❸). Note the above scheme works over both BT and Wi-Fi. When using Wi-Fi, the watch and phone communicate through Google's servers.

We define push notifications as those that trigger ❶ and ❷ in Figure 5. A challenge we face is that they cannot be directly collected so we develop a robust method of extracting them from our data. Recall in Table 2 that we capture all card posts. However, not all cards correspond to push notifications. To capture ❶, we correlate each card posted by the Android Wear service with the network traffic by searching for a specific signature in the traffic that we reverse engineered from controlled experiments. To capture ❷, we also correlate each card posted by a watch-side app with network traffic. This may incur some false positives but we expect the error rate to be very low.

We now describe the results. Table 5 breaks down all push notifications into six categories. As shown, notifications are dominated by instant messaging (IM) and email. We identify more than 10 IM apps and 3 email client apps in the dataset. In particular, *WhatsApp*, a popular IM app, accounts for 23.8% of all notifications. Also, the *Google Quick Search Box* app pushes various notifications such as weather, traffic, news, and stock to the watch. Other apps performing pushes include social media, text message, phone calls, calendar, news, maps, *etc.* Overall, we observe that 230 apps create notifications that are pushed to the watch. Among them, most apps (196 out of 230) do not have a watch-side version (*i.e.,* belong to ❶).

[2]In Android, a *notification* is a message that can be displayed outside of an app's normal UI. It first appears as an icon in the phone's notification area. The user can find its details in the *notification drawer* [1].

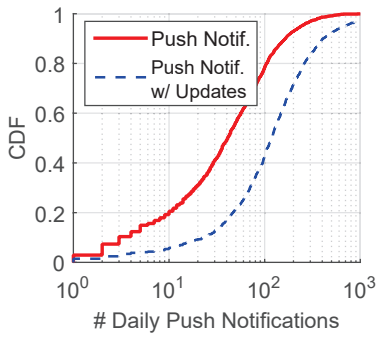


Figure 6: The number of a user's daily received notifications.

The solid curve in Figure 6 plots the distribution of the number of notifications received daily by each user. The number of received notifications are diverse with the 25-th, 50-th, and 75-th percentiles being 14, 41, and 91, respectively. We observe high variation both across users and for the same user across different days. To illustrate this, we compute for each user the average number ($\mu$) and standard deviation ($\sigma$) of notifications across days. Across users, $\mu$ ranges from 10 to 256, and $\sigma$ spans between 13 and 342. This implies the potential challenge for predicting notifications in the long-term. Despite this, we do observe that the arrival of notifications exhibits very strong bursty pattern: the median inter-arrival time of notifications across all users is only 49 seconds, implying potentially good short-term predictability for notifications. This can be explained by the bursty pattern of instant messages that dominate the push notifications. We further observe that if a posted notification card is not removed, it may be constantly updated by the phone-side app. A representative example is turn-by-turn navigation in Google Maps. If such updates are also included in notification counting, the total number will increase by 2.4x, as plotted in the dashed curve in Figure 6.

We have also identified two possible improvements for notifications. First, notifications are often delay-tolerant. Delaying their push and piggybacking them with other notifications or watch wake-up events can lead to significant energy savings, as will be demonstrated in our controlled experiments in §5.3. Second, we find that most (if not all) phone apps today employ very simple logic to determine *whether or not* to push a notification. For example, in its vehicular navigation mode, *Google Maps* keeps sending turn-by-turn updates to the watch – this is unnecessary and may distract the driver if the phone is already mounted to the dashboard for navigation. In another example, *WhatsApp* performs push if and only if the phone-side chat window is not in the foreground. This however may cause both false positives (*i.e.,* the user is interacting with the phone but the message is still pushed to watch[3]) and false negatives (*e.g.,* the user leaves the phone on her desk with the chat window open but the message is not pushed to watch). Ideally, in most use cases, an app should perform pushes only when the user is not interacting with the phone. One possible improvement is thus to introduce an API that intelligently determines whether to push or not to push by leveraging diverse types of information sources such as the user interaction level, the application type, and even the physical distance between the watch and the phone.

[3]The phone will always show the message in its notification area so if the user is looking at the phone, usually there is no need to push.

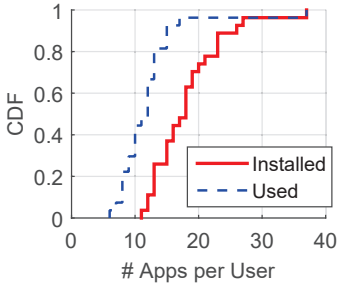


Figure 7: The number of installed vs. used apps per user, across all users.

| Apk Name | Description |
|---|---|
| com.google.android.keep | Note taking |
| com.tencent.mm | WeChat messenger |
| com.google.android.deskclock | Clock |
| com.google.android.apps.fitness | Fitness tracker |
| com.google.android.apps.maps | Map |
| com.callpod.android_apps.keeper | Passwd manager |
| com.appfour.wearmessages | Messenger |
| com.appfour.wearmail | Email |

Table 6: A list of popular apps in our dataset (based on their overall execution duration).

## 4.3 Application Usage

Compared to a traditional watch, a unique feature of a smartwatch is its capability of running diverse 3rd-party applications ("apps"). We first note that as for now, smartwatch apps are still much less popular than smartphone apps: in 10/2016, we crawled the top-500 free and paid apps on Google Play Store, and found only 2.8% of free apps and 3.4% of paid apps have their watch versions. Despite this, a trend we believe is that more and more apps will be shipped with their wearable stubs giving the increasing maturity of the wearable ecosystem [10].

The solid line in Figure 7 plots the distribution of installed apps across users (as of 11/14/2016). An average user has 18 apps installed on her watch. We include built-in apps such as Google Maps but exclude system processes that users cannot interact with. Within the installed apps, how many of them are actually used? The dashed line in Figure 7 plots the distributions of the apps that have been used at least twice. We find that users do use diverse apps on their watches: Table 6 lists the most popular used apps in our dataset; they include messengers, email clients, maps, activity trackers, and note taking software.

Figure 7 also reveals that many installed apps are almost *never* used on watch. This is because a smartwatch app usually has its smartphone counterpart, and both versions (phone-side and watch-side) are bundled together in a single installation package. When the phone version is installed, the watch version will be automatically pushed to and installed on the watch. As a result, a user may only use the app on the phone despite its watch version being automatically installed. We noticed that this issue is addressed in the Developer Preview version of Android Wear 2.0 where the user can choose to only install either the phone-side or the watch-side app [3].

**Activity and Service** are two important components of Android apps. An *activity* represents a visible user interface, whereas a *service* runs in the background without a user interface. Almost all smartphone apps contain one or more activities, and some may run services. Figure 8 plots ranked total running time of activities and services for each watch app in our dataset. We make two

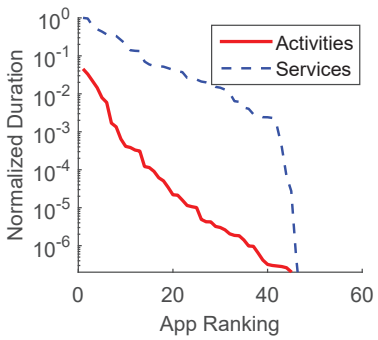


Figure 8: Activities and services ranked by their overall execution duration.

| Duration | $d \leq 1$s | $1 < d \leq 10$s | $10 < d \leq 60$s | $d > 60$s |
|---|---|---|---|---|
| % Services | 71.8% | 12.7% | 5.3% | 10.1% |

Table 7: Distribution of services' duration.

observations. First, despite a large number of lightly used apps, a small fraction of apps (*e.g.,* those listed in Table 6) dominate the usage, a phenomenon also observed for smartphone apps [54]. Second, a more unexpected finding is that the background service execution is 55.8 times longer that of full-screen activities (we filtered out core OS services and the service of our data collector). One may explain this using the common sense that services may continuously run in the background for a long time, as is often the case for smartphone apps. This is however not always the case on Android watches. Table 7 shows the distribution of the lifetime of services. The vast majority of them are very short (less than 1s).

Overall, we find that short but frequent background service executions dominate the Android watch app usage. Based on our controlled experiments, we find many short services are caused by push notification or short message exchanges between watch and phone. We use *WeChat*, a popular instant messenger app, as an example. When the phone pushes a notification containing a new message to the watch, the watch will be woken up, and a notification card will be displayed (Figure 9 left, following ❷ in Figure 5). There are several possible actions that a user can then take: *(a)* take a glance of the message and ignore it; *(b)* open the *WeChat* app on her phone to reply to the message; and *(c)* directly reply to the message on her watch by swiping the card and tapping "reply"; in that case the watch version of the *WeChat* app (with an activity) will be launched. As indicated by the short notification-triggered wake-up session duration shown in Table 4, users seldom take Action *(c)*, very likely due to the watch's small form factor making full-fledged user interaction difficult. As shown in the right side of Figure 9, The watch version of *WeChat* only allows replying to a message by voice, sticker icons, or pre-defined text. On the other hand, Action *(a)* and *(b)* only involve short-lived services. For each notification of *WeChat*, we observe three ephemeral services invoked by *WeChat* and some other short-lived services started by the system processes managing the card and push notification. We observe similar patterns for other popular apps.

# 5. ENERGY CONSUMPTION

Smartwatches have very limited battery capacity. Our LG Urbane watch has a battery of only 410 mAh, much smaller than that of a typical smartphone battery (2K to 3K mAh). Charging a watch requires special charging dock making it difficult for users to charge the watches during the day. Understanding watches' battery usage can greatly facilitate the development of new power management solutions.


Figure 9: **Left**: a WeChat card showing at the bottom of the screen. **Right**: reply to a message using the WeChat app on watch.

## 5.1 Smartwatch Power Models

A prerequisite for fine-grained energy analysis is a *power model*, which is a function $E(\vec{A})$ that maps $\vec{A}$, system activities and events directly measurable on the device, to their incurred energy and power consumption. In the literature, numerous studies have derived energy models for smartphones [55, 47, 26, 42, 41, 38, 16]. Nevertheless, to our knowledge, no power model is publicly available for smartwatches whose energy consumption profiles are quite different. To fill this gap, we empirically derive accurate power models for today's off-the-shelf smartwatches.

To measure the watch's energy, we extract a compatible battery interface, which is used as a connector between the watch and the Monsoon power monitor [5], from a same-vendor smartphone. Our modeling approach follows the high-level methodology for smartphone power modeling [55, 16, 38]. When measuring a component, we keep other components offline (*e.g.,* Wi-Fi, BT, display) or at a steady power state (*e.g.,* CPU) whose power consumption is then subtracted from measured power value. For components involving parameters (*e.g.,* CPU utilization), we programmatically change them and use regression to derive an empirical model as a function of the parameters. We repeat each experiment 10 times and use the average power for modeling. The overall watch power is then estimated as the sum all components' power consumption.

We study two commercial smartwatches: LG Urbane (used in our user study) and LG Watch R. LG Watch R has a similar configuration compared to LG Urbane except that it does not have built-in Wi-Fi. Table 8 presents our derived power models for the two watches. We first highlight our findings for the LG Urbane watch. The power consumption of each state (awake, dozing, and sleeping) consists of an almost-constant base power plus power consumption of other components such as CPU, display, and network interfaces. (1) At the dozing state, the overall device power consumption is low (24.3 mW). The watch face display power accounts for about half of the overall dozing power. Since the watch face display has low brightness and mostly dark colors, its power can be modeled using a constant value (about 12.2 mW). (2) At the sleeping state, the power consumption is even lower since the display is turned off. Also note that the watch is equipped with a low-power movement sensor and a microphone, whose power consumption is included in the base power. Since they run in an "ambient" manner (*i.e.,* always-on), it is difficult for us to separate their individual power contributions. (3) The watch is equipped with a 1.3 inch 320x320 P-OLED display, whose power is determined by the brightness level and the pixel colors [17, 19] when the watch is awake. We find that blue is the most energy-consuming color component, followed by green and then red. Note our model does take into account the circular shape of

| Component | Power Consumption (mW) | |
|---|---|---|
| | LG Urbane (user study) | LG Watch R |
| Dozing base + display | 24.3 (base 12.1 + display 12.2) | 22.7 (base 10.7 + display 12.0) |
| Sleeping base | 14.5 | 17.6 |
| Wake-up base | 47.1 | 54.9 |
| CPU | $214.0u$, $u \in$ (0,1]: CPU util. | $216.2u$, $u \in$ (0,1] |
| Wake-up display power = $\sum(c_r \cdot r + c_g \cdot g + c_b \cdot b + C)/K$, Per-pixel $r, g, b \in$ [0, 255], K = 320*320. Values of $\{c_r, c_g, c_b, C\}$ are listed below. | | |
| Wake-up brightness 1 | {.023, .060, .084, 67.2} | {.014, .036, .092, 60.6} |
| Wake-up brightness 2 | {.034, .071, .129, 67.2} | {.027, .060, .126, 60.6} |
| Wake-up brightness 3 | {.041, .092, .167, 67.2} | {.029, .077, .159, 60.6} |
| Wake-up brightness 4 | {.055, .120, .201, 67.2} | {.044, .096, .210, 60.6} |
| Wake-up brightness 5 | {.058, .144, .236, 67.2} | {.065, .127, .255, 60.6} |
| Wake-up brightness 6 | {.076, .179, .303, 67.2} | {.077, .163, .325, 60.6} |
| BT Tail | 4.77 sec, Power: 34.1 | 4.00 sec, Power: 13.88 |
| BT Data ($\sim$ 0.5m) | Tx: 111.5, Rx: 117.2 | Tx: 103.0, Rx: 104.6 |
| BT Data ($\sim$ 5m) | Tx: 132.9, Rx: 116.2 | Tx: 121.5, Rx: 97.3 |
| BT Data ($\sim$ 10m) | Tx: 130.8, Rx: 113.9 | Tx: 120.9, Rx: 98.2 |
| BT Scan | 146.0 | 155.1 |
| Screen touch/swipe | 198.9 | 182.3 |

Table 8: Derived power models for LG Watch Urbane and LG Watch R.

| Component | Power (mW) |
|---|---|
| Wi-Fi Tail | 0.18 sec, Power: 121.2 |
| Wi-Fi Promotion | 0.30 sec, Power: 242.5 |
| Wi-Fi Data (-42 dBm) | Tx: 669.1, Rx: 378.5 |
| Wi-Fi Data (-55 dBm) | Tx: 672.8, Rx: 343.0 |
| Wi-Fi Data (-65 dBm) | Tx: 840.7, Rx: 341.8 |
| Wi-Fi Scan | 252.3 |

Table 9: Wi-Fi power model for LG Urbane.

| Use Case | Trace Length | Energy Consumption ($\mu$Ah) | | |
|---|---|---|---|---|
| | | Measure | Model | Err. |
| Check time | 11 sec | 206 | 196 | 4.7% |
| Google map over BT | 60 sec | 1881 | 1792 | 4.7% |
| Fitness tracking | 30 sec | 773 | 730 | 5.5% |
| Web browsing over Wi-Fi | 60 sec | 2407 | 2513 | 4.4% |
| Hangout Chat over BT | 30 sec | 893 | 876 | 1.9% |
| Push notify. over BT | 65 sec | 1378 | 1437 | 4.3% |
| Push notify. over Wi-Fi | 65 sec | 1933 | 1920 | 0.7% |

Table 10: Power model validation for LG Urbane.

the watch display [36] since only the displayed pixels are counted. (4) The CPU power is determined by three factors: the number of online cores, the frequency of each core, and the utilization of each core. The LG Urbane watch is equipped with a quad-core Qualcomm Cortex A7 processor. However, three of the cores are forced to be offline by the OS, and the clock of the only online core is fixed at 787 Mhz. This is a common practice on Android smartwatches [32]. Therefore, the only factor affecting the power is the CPU utilization, and we find both are linearly correlated. (5) The Bluetooth state machine consists of an idle and an active state. The state promotion takes negligible time, while the demotion from the active to the idle state is triggered by an inactivity timer ("tail time") of 4.77s. (6) We find that the capacitive touch input also incurs non-negligible power consumption. (7) Table 9 measures the Wi-Fi power model for LG Urbane. The Wi-Fi state machine is similar to that of smartphones [16, 38], except that we observe a non-trivial state promotion delay of 0.3s. Also, we find when the RSSI is lower than -70dBm, likely due to its small built-in antenna, the watch has difficulty associating with the AP. For LG Watch R, as shown in Table 8, its power model is qualitatively similar to that of LG Urbane except that it does not have a Wi-Fi interface.

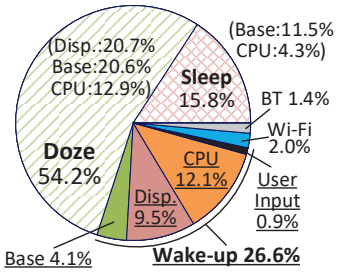


Figure 10: Component-level energy breakdown across all users.

**Power Model Validation.** We conduct thorough validation for our models for both watches at the component level and device level. Table 10 shows the in-lab device-level validation results for LG Urbane. We study seven diverse use cases. The "measure" and "model" columns correspond to the device-level energy consumption measured from the power monitor and computed from our model, respectively. The error rates, defined as $|E(\text{Model}) - E(\text{Measure})|/E(\text{Measure})$, are less than 6%. For LG Watch R, the device-level error rates are less than 5% for these usage scenarios (table not shown).

## 5.2 Energy Consumption in the Wild

We now apply the LG Urbane power model to our user study dataset, to quantify the energy consumption of smartwatches in realistic settings. Our dataset contains all information needed by the power model except the following limitations. First, we are not able to obtain the BT signal strength using Android Wear API, so we use the 0.5m BT power model. Second, to ensure the data collector itself incurs very low overhead, we capture a screenshot every 30 seconds (§3). Such an interval may lead to less accurate display energy estimation (in our validation we use an interval of 1 second). Note this is already more accurate than prior smartphone display modeling approaches that do not consider the displayed content [14] or sample the pixels every several minutes [55]. We also want to remind readers that the results below are relevant to the specific brand of the watch used in our study (LG Urbane), and other types of watches may possibly exhibit different energy consumption profiles.

Recall that Table 3 shows the energy breakdown among the three states: awake, dozing, and sleeping. More than half of

the overall energy is consumed by the dozing state due to its long duration despite its low power. But somewhat unexpectedly, despite the very short duration of the awake state (2%), its energy contribution is as high as 27.2%. Figure 10 shows a more fine-grained energy breakdown. Within the dozing state, the energy consumed by the base, display, and CPU are roughly 1.6:1.6:1. Despite its low brightness, display is still a key factor of battery drain when the watch is dozing. The dozing CPU energy comes from the maintenance window (§4) that periodically wakes up the CPU. Within the awake state, display and CPU also dominate the energy consumption, accounting for 34.9% and 44.5% of the wake-up energy, respectively (or 9.5% and 12.1% of the overall energy respectively). The network incurs very small energy footprint (3.4%). About 22% of the Wi-Fi energy and 9% of the Bluetooth energy are spent at the awake state (not shown in Figure 10).

From Table 8 and Figure 10, we compute the average power consumption at the wakeup, dozing, and sleeping state to be 309.8mW, 31.9mW, and 14.5mW, respectively, across all users. Note that the average dozing power (31.9mW) includes the base, the display, and the CPU power. By weighing them using the usage duration breakdown listed in Table 3, we can further compute the smartwatch's average power consumption to be 25.9mW across all states and users. Assuming that, a fully charged watch (with a battery capacity of 410 mAh for LG Urbane) can last for about 41.7 hours.

**Comparison with Smartphone.** We next compare our results with smartphone energy consumption profiles reported in the literature. We pick [14], a recent crowd-sourced smartphone energy study, to facilitate the discussion. We highlight three observations. First, the authors of [14] report that on a smartphone, the top two energy consumers are CPU (30.6%) and display (27.4%). This is also true for smartwatches: CPU accounts for 29.3% of the overall energy; more interestingly, despite smartwatches' small screen sizes, display still contributes to 30.2% of the overall energy. Second, networking is very power-hungry on smartphones: cellular and Wi-Fi consume 26.4% and 8% of the overall energy, respectively. But networking (Wi-Fi and Bluetooth) is responsible for only 3.4% of the overall smartwatch energy. Besides an apparent reason that most watches do not have a cellular interface, such a disparity is also caused by the different traffic patterns between a watch and phone as well as the fact that Bluetooth dominates the smartwatch network usage, as to be studied in §6. Third, [14] reports that an average smartphone spends about 2 hours every day on the screen-on state, which incurs about 59% of the overall energy consumption. The smartwatch case is very different: an average user's watch wakes up for only about 10 minutes every day, corresponding to about 27% of the energy (Table 3). Meanwhile, smartwatches are more energy-efficient in non-awake (*i.e.,* dozing and sleeping) states where user application and system activities are strictly restricted (§4).

**Diversity across Users.** We equally divide the 27 users into three groups: light ("L"), medium ("M"), and heavy users ("H"), based on users' average daily awake duration. The left side of Figure 11 plots the duration breakdown for each group where "W", "S+D", and "C" correspond to awake, sleeping/dozing, and charging, respectively. The three groups exhibit similar patterns except that the heavy users spend a little more time on the awake and charging states. The right side of Figure 11 plots the corresponding energy breakdown, where the wake-up energy consumption is significantly higher for heavy users. The results again indicate that smartwatch's active usage time is unanimously short across users. However, due to the big power consumption gap between awake and dozing/sleeping, a small increase of the wake-up duration will be "amplified" from the battery draining perspective, thus shortening the overall standby time of the watch.

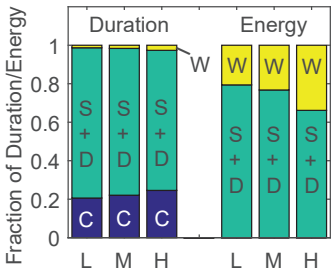


Figure 11: Usage duration and energy consumption diversity across three user groups: **L**ight, **M**edium, and **H**eavy users.

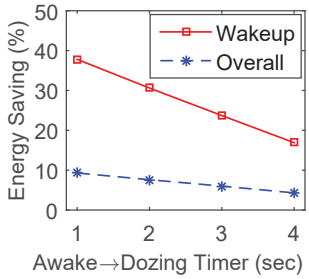


Figure 12: Tuning the Awake → Dozing Timer.

## 5.3 Improving Smartwatch Energy Efficiency

We study five methods for improving the smartwatch energy efficiency. Note that although some of them have been applied to smartphones, our contributions here are two-fold: we study them in the context of smartwatches; we also perform "what-if" analysis on our dataset to reveal their impact on real smartwatch workloads (or doing controlled experiments if a trace-driven analysis is not feasible).

**1. Tuning the Awake→Dozing Timer.** We consider tuning the inactivity timer that controls the transition from awake to dozing. Recall in Figure 1 that its default value is 5 seconds. We develop a trace-driven simulator that takes as input the original user study trace, the smartwatch power model, and a new timer setting. The simulator computes the energy consumption in the new setting: if the timer is reduced to $x < 5$ seconds, we turn the last $5-x$ seconds within an original wake-up session into dozing by changing its power consumption accordingly. Figure 12 plots the energy reduction in the "what-if" scenario where the timer is set to 1 to 4 seconds. Doing so leads to energy savings between 17% and 38% for the awake state. The overall energy saving, however, ranges from only 4% to 9%, which is equivalent to 1.7 to 3.8 hours of average usage time (assuming 41.7 hours of battery capacity as computed in §5.2). Note that a more intelligent approach the OS may take is to dynamically determine when to doze. This has already been partially implemented in Android Wear 2.0, which uses user's gesture (*i.e.,* moving the wrist away) as a signal to dim the watch in addition to using a regular timer. Although more sensing techniques can be developed (*e.g.,* dimming the watch when it is covered by a sleeve), the overall energy saving brought by this scheme is limited.

**2. Improving the Dozing→Sleeping Mechanism.** Recall in Figure 1 that the current policy adopted by Android Wear is to enter the sleeping state after an idle period of 35 minutes only when the watch is not worn. We consider a more energy-efficient scheme

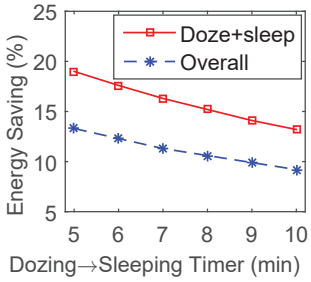


Figure 13: Sleep after a timeout even when the watch is worn.

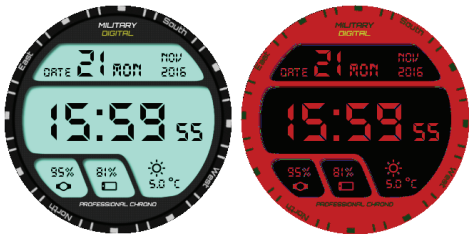

Figure 14: An example of power-saving color transformation.

where even when the watch is worn, it enters the sleeping state after a timeout. Our dataset suggests that almost all (98%) dozing sessions (including those when the watch is worn) are shorter than 35 minutes, implying the current timeout is too conservative if applied to our scheme. In Figure 13, we simulate our new scheme by applying a shorter dozing→sleeping timer (regardless of whether the watch is worn or not) to our dataset. On one hand, changing the timeout to 5 to 10 minutes brings no effect on most (80% to 88%) dozing sessions, so the impact on the user experience is marginal: when the user looks at the watch face, it is still on as before. On the other hand, this very simple tuning leads to a decent amount of energy savings: 13% to 19% of the non-awake energy or 9% to 13% of the global energy (equivalent to 3.8 to 5.4 hours of average watch usage time).

**3. Power-saving Color Transformation.** A wide range of smartwatches today use OLED display where different colors incur different power consumption. We apply the power-saving color transformation technique that was proposed in [18] and was applied to smartphone web browsing in [19]. Its high-level idea is to map a display's color histogram to a more energy-efficient one while satisfying human perception constraints in a color space (CIELAB) designed to mimic human vision. We apply the algorithm (using the "Color64" scheme in [18]) to all wake-up screenshots in our dataset, and find that for most screens, the algorithm yields recognizable display despite the colors becoming less perceptually pleasing to humans (an inherent limitation of this approach as illustrated in Figure 14). Nevertheless, color transformation results in 41.4% reduction of the overall wake-up display energy across all users. This provides incentives to develop energy-efficient user interfaces for wearables with OLED display.

**4. Bundling Delay-tolerant Push Notifications.** Recall in §4.2 that many push notifications are delay-tolerant. We conduct controlled experiments to quantify how much energy can be potentially saved when delay-tolerant notifications are bundled together. We wrote a phone-side app that performs push using Method ❶ in Figure 5. The app employs three schemes ($S_1$ to $S_3$) to push $n$ notifications, each containing a short message, in $t$=5 minutes: in $S_1$, the notifications are equally spaced with inter-arrival time (IAT) of $t/(n-1)$; (2) in $S_2$, they are pushed one-by-one with an IAT of 3 seconds to allow the user to read each message; (3) in $S_3$, all messages are packed into a single notification pushed to the watch. Table 11 lists the normalized overall energy consumption for the three push schemes with $n$ set to 5, 10, and 20. The results suggest that bundling effectively decreases the energy consumption by reducing the overall wake-up period. Designing a full-fledged delay-tolerant-aware push notification system for wearables is our future work.

| | $S_1$ | $S_2$ | $S_3$ |
|---|---|---|---|
| $n=5$ | 100% | 74% | 46% |
| $n=10$ | 100% | 66% | 34% |
| $n=20$ | 100% | 58% | 19% |

Table 11: Normalized energy consumption when bundling delay-tolerant push notifications. $n$ is the number of notifications and $S_1$ to $S_3$ are three delivery schemes.

| CPU Config. | $C_1$ | $C_2$ | $C_3$ | $C_4$ |
|---|---|---|---|---|
| # Cores | 1 | 1 | 2 | 2 |
| Freq. each core | 787 Mhz | 384 Mhz | 787 Mhz | 384 Mhz |
| $W_1$: app launch | 265, 7.6s | 278, 9.8s | 175, 4.4s | 161, 4.5s |
| $W_2$: push notif. | 46, 1.8s | 64, 2.9s | 61, 1.8s | 49, 1.9s |

Table 12: Balancing the energy-QoE tradeoff for two workloads using four CPU configurations. In each cell, the first number is the energy consumption (in $\mu$Ah), and the second number is the QoE. All cells are averaged over 3 runs with standard deviation $< 10\%$.

**5. Workload-aware CPU Configuration.** Recall in §5.1 that on a quad-core watch, Android Wear forces three cores offline and fixes the frequency of the only online core. We conduct controlled measurements to explore the performance-energy tradeoff incurred by different CPU configurations under diverse workloads. We consider four CPU configurations: $C_1$ uses one core at the default frequency (787 Mhz); $C_2$ employs one core at a lower frequency (384 Mhz); $C_3$ and $C_4$ use two cores at 787 Mhz and 384 Mhz, respectively. We study their performance under two common workloads: $W_1$ for launching the *Google Maps* app and $W_2$ for receiving/posting a push notification. Table 12 measures the (Energy, QoE) for the 8 combinations of CPU configuration and workload. The energy consumption is measured by the power monitor at the device level, and the QoE metric depends on the workload: app launch time for $W_1$ and the notification delivery time for $W_2$ (from the reception of the first byte to the notification being posted). As shown, the CPU configuration yielding the best energy-QoE tradeoff depends on the workload: $C_4$ for $W_1$ and $C_1$ for $W_2$. The results imply that using multiple cores can potentially improve both the QoE and the energy efficiency for common smartwatch workloads such as app launching.

# 6. NETWORK TRAFFIC

Many smartwatches (including our user study watches) are equipped with Bluetooth and Wi-Fi interfaces. Per Android Wear OS, if the watch is paired with a phone over Bluetooth (BT), Wi-Fi is turned off for saving energy and the watch uses the phone as a "gateway" to access the Internet. If the phone is not present, the watch can directly access the Internet using TCP/IP over Wi-Fi. We now provide a first-of-its-kind measurement of smartwatch traffic. Our characterizations provide hints for improving the network stack for future wearable devices.

We first answer an important question: how often does the watch pair with a phone? Using our dataset, we find that during most of the time (80%), the watch and phone are paired. This is somewhat expected as we usually carry phones with us. During

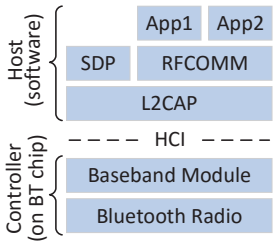


Figure 15: The BT protocol stack of Android smartwatch.

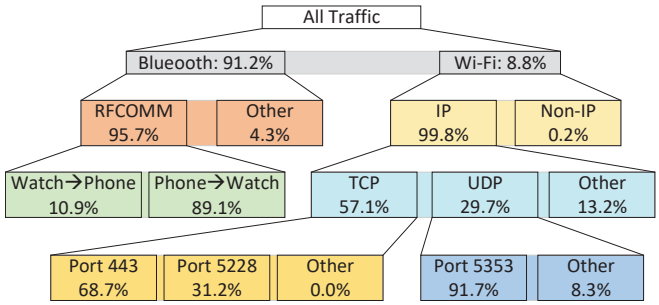


Figure 16: Smartwatch traffic volume breakdown.

the daytime (9am to 9pm), the percentage is slightly higher (84%), likely because some users turn off their phones at night.

Before presenting detailed characterization, we give some background of the BT protocol stack used by Android smartwatches[4] shown in Figure 15 [23, 13]. The *controller* resides on a BT chip, which performs lower-layer functions such as (de)modulation, physical channel management, and peer discovery. The *host* realizes higher-layer protocols in software (OS or device driver). The Logical Link Control and Adaptation Protocol (L2CAP) provides multiplexing of upper layer protocols such as SDP and RFCOMM. The Service Discovery Protocol (SDP) allows to discover services provided by nearby BT devices. RFCOMM [12] is the data-plane protocol allowing multiple application streams to be multiplexed. Between the controller and host lies the Host-Controller Interface (HCI), which is our data collection point. We build a tool (in ∼1,300 LoC) that processes the collected BT traces. It parses HCI signaling messages and L2CAP/SDP/RFCOMM packets, and outputs assembled and demultiplexed RFCOMM streams carrying the application data. Note non-smartwatch devices may use different upper-layer protocols over BT.

**BT Traffic.** Figure 16 shows the overall traffic volume breakdown. Since smartwatches are usually paired with phones, the vast majority (91.2%) of bytes are delivered over BT. Most BT traffic belongs to the RFCOMM protocol that carries user data. Also downlink BT traffic (from phone to watch) overweighs the uplink, as a smartwatch is mostly a consumer device that receives data from the phone.

We next investigate three key characteristics of BT traffic: flow size, duration, and rate, which are known to be the most important metrics for Internet flows [56, 45]. One issue we need to address is how to define a *flow* for BT. Since RFCOMM streams are usually persistent and long-lived (they may last for tens of minutes or even hours), we employ an idle period threshold $i$ to split a (demultiplexed) RFCOMM stream into segments, such that the packet inter-arrival time (IAT) within a segment is shorter than $i$ and IAT across segments is at least $i$. We then regard each segment to be a user flow.

[4]We verified three Android watches: LG Urbane, LG Watch R, and Samsung Gear. They all use the same protocol stack in Figure 15.

| Idle gap threshold $i$ | 5 sec | 10 sec | 20 sec |
|---|---|---|---|
| 1. Flows initiated by phone | 72.8% | 68.4% | 71.0% |
| 2. Flows initiated by watch w/o UI | 24.7% | 28.0% | 26.0% |
| 3. Flows initiated by watch w/ UI | 2.5% | 3.6% | 3.0% |

Table 13: Origin of BT traffic (breakdown by bytes).

Figure 17 and 18 plot the distributions of BT flow size and duration, respectively. The vast majority of flows are very small and short compared to smartphone traffic [20, 27]: when the idle threshold $i$=10 sec, 76.8% of the flows are smaller than 10KB, and 53.1% of the flows are shorter than 1 sec. Only 0.4% of the flows are considered to be indeed large ($>$ 1MB), and only 0.1% are indeed long-lived ($>$ 100 sec). Based on manual inspection of the payload of these "heavy-hitter" flows, we identify some of their semantics to be the following: app download, web browsing (some users do use the on-watch browser to surf the Internet), continuous data download (*e.g.,* map tiles), and synchronization of other large data. Figure 17 and 18 also indicate that the selection of the idle threshold $i$ has very small impact on the measured distributions.

Figure 19 measures the BT flow rate, defined as flow size divided by duration, using $i$=10 sec. In order for the computed rate to be meaningful, we only include flows whose duration is at least $d \in \{0.5, 1.0 \text{ sec}\}$. Also downlink (phone to watch) and uplink (watch to phone) rates are measured separately. As shown, the vast majority of flows are slow, and uplink flows are even slower than downlink: only 3.5% (21.5%) of uplink (downlink) flows are faster than 100kbps. We also find that flow sizes and rates are highly correlated. When $d$=1 sec, the Pearson Correlation Coefficient between $log(\text{size})$ and $log(\text{rate})$ is 0.84 and 0.88 for uplink and downlink, respectively, much higher than those for Internet flows [45].

We next investigate the origin of BT traffic by answering the following question: how much BT traffic is triggered by a user? We break down all BT flows into three categories: (1) a flow is initiated from the phone side, as indicated by the downlink direction of the first data packet within the flow, (2) a flow is initiated from the watch side, as indicated by the uplink direction of the first packet, and the network activity is *not* triggered by user interaction[5], and (3) a flow is initiated from the watch side by user interaction. As shown in Table 13, most of BT traffic belongs to Category 1 (*e.g.,* data/notification push); a small fraction of the traffic belongs to Category 2 (*e.g.,* periodical fitness data upload), and only a tiny fraction of traffic falls into Category 3 (*e.g.,* user touches the watch and triggers some traffic). Note within Category 1, we are not able to tell the fraction of traffic initiated by a user interacting with the *phone*. Nevertheless we expect that fraction to be low, because most of the pushes and background sync are indeed "spontaneously" incurred.

**Wi-Fi Traffic** plays a less important role in smartwatch. As shown in Figure 16, more than half of the Wi-Fi traffic is TCP, which is dominated by server port 443 and 5228. For port 443, most server IPs point to Google that provides various services such as push notification. Port 5228 is used by Google Play Store. Surprisingly, due to Google's "HTTPS everywhere" principle, we do not observe any port 80 traffic, which still prevails in the smartphone world today. It is therefore difficult for a middlebox to use DPI-based approach to identify Android

[5]We use a 2-second window [$t_0$-2, $t_0$] to detect user interaction right before a flow where $t_0$ is the timestamp of the first data packet of a flow. Different window sizes yield qualitatively similar results.

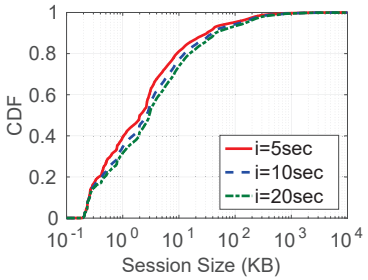

Figure 17: BT flow size distributions.

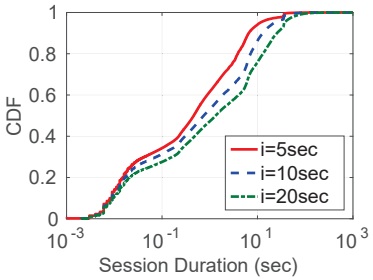

Figure 18: BT flow duration distributions.

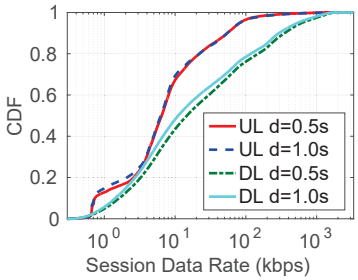

Figure 19: BT flow rate distributions ($i$=10s).

smartwatch traffic. Regarding UDP, more than 90% of the UDP traffic is for name resolution (port 53/5355/5353), and the remaining UDP traffic mostly relates to DHCP (port 67) and SSDP (port 1900). Virtually no UDP traffic carries application data. Wi-Fi flows exhibit characteristics similar to those of BT flows observed in Figure 17, 18, and 19.

**Handover between BT and Wi-Fi.** Dual networks are common on mobile devices. We are interested in how smartwatches handle the handover between BT and Wi-Fi, a common use case that needs to be supported when the watch moves out of or into the phone's BT coverage. We find that many smartwatch apps do support handover, but oftentimes a handover takes a long time (more than 5 seconds). This may be unacceptable for real-time streaming between the watch and phone (*e.g.,* Samsung Gear S3 Frontier has a speaker, making it possible to answer phone calls or to do VoIP on the watch). Researchers have proposed robust and transparent handover solutions such as Multipath TCP (MPTCP [21]). However, since BT by default does not speak TCP/IP, it is difficult to directly apply MPTCP on smartwatches. The potentially excessive power consumption of maintaining multiple interfaces simultaneously is also a concern. More research needs to be done in this direction.

## 7. DISCUSSION

**Limitations.** We discuss several limitations of our current user study deployment.

• First, our study lacks device diversity since all participants use the same type of watch. Although we use a popular commodity watch (LG Urbane) with state-of-the-art hardware, we admit that smartwatches are diverse, from basic ones with black-and-white display (*e.g.,* Pebble Classic) to advanced models with built-in GPS and cellular (*e.g.,* Samsung Gear S3 Frontier). Additional hardware components such as GPS and cellular bring in new applications, and may thus change user behaviors and energy consumption profiles of smartwatches. We however believe that due to the very nature of smartwatches (wearable, small form factors, *etc.*), the basic usage patterns such as short "flick and look" wake-up sessions and key applications such as push notification reception tend to remain the same.

• Second, as mentioned in §4, the participants in our study have limited diversity (faculty, students, and staff members at Indiana University). We plan to further increase the user base and its diversity in our future deployment.

• Third, there are other factors that might impact the user behavior and thus the measurement results. For example, our participants do not own the watches (in contrast, in [44], users used their personal smartwatches). In another example, the participants were given similar orientation sessions making them "equally" familiar with the device. The distribution of feature usages may possibly be different for "untrained" users than a set of users who had a good orientation.

• Lastly, since all our data collection is performed on the watch, we are not able to obtain information that is only collectible on the phone. For example, it is impossible for us to tell how a smartwatch impacts the smartphone usage, or to quantify the impact of a watch on its paired phone's energy consumption. We plan to expand the data collector to phones in our future work.

**Other Smartwatch OSes.** The OS also plays an important role in determining the resource consumption [32]. In this study we focus on Android Wear Version 1.3 while other wearable OSes such as Apple watchOS and Tizen exist. We are in particular interested in Android Wear 2.0 that was in its developer preview stage when this paper was written. Android Wear 2.0 provides several new features that enhance the usability and energy-efficiency of wearable devices. To name a few, it supports standalone wearable apps that do not require their smartphone counterparts and that have direct Internet access through Wi-Fi or cellular; it allows the transition to the dozing state through a dynamic timer; it introduces "Complications API [2]" to allow 3rd-party apps to show custom data on a watch face; it also has new input methods including built-in virtual keyboards. We plan to conduct an in-depth study of Android Wear 2.0 in our future work.

## 8. CONCLUDING REMARKS

Through an IRB-approved user study involving 27 users for 106 days, we conducted an in-depth analysis of smartwatch usage in the wild by answering several key research questions such as how users interact with the watch, how smartwatch apps behave, how push notifications look like on watches, how smartwatch energy is consumed, and what key characteristics of smartwatch traffic are. Our findings shed light on improving the design for wearable systems, which indeed requires cross-layer efforts and involves multiple entities in the wearable ecosystem.

## Acknowledgements

We would like to thank the smartwatch users at Indiana University Bloomington who voluntarily participated in our user study. The user study devices were sponsored by Indiana University Bloomington. We also thank the MobiSys reviewers and especially Elizabeth M. Belding for shepherding the paper. Feng Qian's research was supported in part by NSF Award #1629347. Felix Xiaozhu Lin was supported in part by NSF Award #1464357 and a Google Faculty Award. Kai Chen was supported in part by NSFC U1536106, 61100226, Youth Innovation Promotion Association CAS, and strategic priority research program of CAS (XDA06010701).

# 9. REFERENCES


[1] Android Notifications. https://developer.android.com/guide/topics/ui/notifiers/notifications.html.

[2] Android Watch Face Complications. https://developer.android.com/wear/preview/features/complications.html.

[3] Android wear 2.0 developer preview 3: Play store and more. http://android-developers.blogspot.com/2016/09/android-wear-2-0-developer-preview-3-play-store-and-more.html.

[4] Apple watch human interface guidelines. https://developer.apple.com/watchos/human-interface-guidelines/overview/.

[5] Monsoon Power Monitor. https://www.msoon.com/LabEquipment/PowerMonitor/.

[6] Optimizing for Doze and App Standby. https://developer.android.com/training/monitoring-device-state/doze-standby.html.

[7] PhoneLab: A Smartphone Platform Testbed. https://phone-lab.org/.

[8] Samsung Gear Application Programming Guide. http://img-developer.samsung.com/contents/cmm/Samsung_Gear_Application_Programming_Guide_1.0.pdf.

[9] Sending and Syncing Data with Watch. https://developer.android.com/training/wearables/data-layer/index.html.

[10] Smartwatches now more popular than Swiss watches. https://www.cnet.com/news/smartwatches-now-more-popular-than-swiss-watches-thanks-largely-to-apple/.

[11] UI patterns for android wear. https://developer.android.com/design/wear/patterns.html.

[12] RFCOMM WITH TS 07.10. https://www.bluetooth.org/docman/handlers/DownloadDoc.ashx?doc_id=40909, 2012.

[13] Specification of the Bluetooth System (Core Specification). https://www.bluetooth.com/specifications/adopted-specifications, 2014.

[14] CHEN, X., DING, N., JINDAL, A., HU, Y. C., GUPTA, M., AND VANNITHAMBY, R. Smartphone Energy Drain in the Wild: Analysis and Implications. In *SIGMETRICS* (2015).

[15] CHEN, X., GROSSMAN, T., WIGDOR, D., AND FITZMAURICE, G. Duet: Exploring joint interactions on a smart phone and a smart watch. In *ACM CHI* (2014).

[16] CHEN, X., JINDAL, A., DING, N., HU, Y. C., GUPTA, M., AND VANNITHAMBY, R. Smartphone Background Activities in the Wild: Origin, Energy Drain, and Optimization. In *MobiCom* (2015).

[17] DONG, M., CHOI, Y.-S. K., AND ZHONG, L. Power Modeling of Graphical User Interfaces on OLED Displays. In *DAC* (2009).

[18] DONG, M., CHOI, Y.-S. K., AND ZHONG, L. Power-Saving Color Transformation of Mobile Graphical User Interfaces on OLED-based Displays. In *ISLPED* (2009).

[19] DONG, M., AND ZHONG, L. Chameleon: A Color-Adaptive Web Browser for Mobile OLED Displays. In *MobiSys* (2011).

[20] FALAKI, H., MAHAJAN, R., KANDULA, S., LYMBEROPOULOS, D., AND ESTRIN, R. G. D. Diversity in Smartphone Usage. In *Mobisys* (2010).

[21] FORD, A., RAICIU, C., HANDLEY, M., AND BONAVENTURE, O. TCP Extensions for Multipath Operation with Multiple Addresses. RFC 6824, 2013.

[22] GOUVEIA, R., KARAPANOS, E., AND HASSENZAHL, M. How do we engage with activity trackers? a longitudinal study of habito. In *UbiComp* (2015).

[23] GUPTA, N. Inside Bluetooth Low Energy. Arech House, 2013.

[24] HAM, M., DAE, I., AND CHOI, C. Lpd: Low power display mechanism for mobile and wearable devices. In *Usenix ATC* (2015).

[25] HUANG, J., BADAM, A., CHANDRA, R., AND NIGHTINGALE, E. B. Weardrive: Fast and energy-efficient storage for wearables. In *USENIX ATC* (2015).

[26] HUANG, J., QIAN, F., GERBER, A., MAO, Z. M., SEN, S., AND SPATSCHECK, O. A Close Examination of Performance and Power Characteristics of 4G LTE Networks. In *Mobisys* (2012).

[27] HUANG, J., QIAN, F., GUO, Y., ZHOU, Y., XU, Q., MAO, Z. M., SEN, S., AND SPATSCHECK, O. An In-depth Study of LTE: Effect of Network Protocol and Application Behavior on Performance. In *SIGCOMM* (2013).

[28] HUANG, J., XU, Q., TIWANA, B., MAO, Z. M., ZHANG, M., AND BAHL, P. Anatomizing Application Performance Differences on Smartphones. In *Proc. ACM MobiSys* (2010).

[29] LAZAR, A., KOEHLER, C., TANENBAUM, J., AND NGUYEN, D. H. Why we use and abandon smart devices. In *UbiComp* (2015).

[30] LIKAMWA, R., WANG, Z., CARROLL, A., LIN, F. X., AND ZHONG, L. Draining our glass: An energy and heat characterization of google glass. In *APSys* (2014).

[31] LIU, R., JIANG, L., JIANG, N., AND LIN, F. X. Anatomizing System Activities on Interactive Wearable Devices. In *APSys* (2015).

[32] LIU, R., AND LIN, F. X. Understanding the Characteristics of Android Wear OS. In *MobiSys* (2016).

[33] LIU, X., ZHOU, Z., DIAO, W., LI, Z., AND ZHANG, K. When good becomes evil: Keystroke inference with smartwatch. In *CCS* (2015).

[34] LYONS, K. What can a dumb watch teach a smartwatch? informing the design of smartwatches. In *ISWC* (2015).

[35] MAYBERRY, A., HU, P., MARLIN, B., SALTHOUSE, C., AND GANESAN, D. ishadow: Design of a wearable, real-time mobile gaze tracker. In *MobiSys* (2014).

[36] MIAO, H., AND LIN, F. X. Tell your graphics stack that the display is circular. In *HotMobile* (2016).

[37] MIN, C., KANG, S., YOO, C., CHA, J., CHOI, S., OH, Y., AND SONG, J. Exploring current practices for battery use and management of smartwatches. In *ISWC* (2015).

[38] NIKA, A., ZHU, Y., DING, N., JINDAL, A., HU, Y. C., ZHOU, X., ZHAO, B. Y., AND ZHENG, H. Energy and Performance of Smartphone Radio Bundling in Outdoor Environments. In *WWW* (2015).

[39] NIKRAVESH, A., YAO, H., XU, S., CHOFFNES, D., AND MAO, Z. M. Mobilyzer: An Open Platform for Controllable Mobile Network Measurements. In *Mobisys* (2015).

[40] NIRJON, S., GUMMESON, J., GELB, D., AND KIM, K.-H. Typingring: A wearable ring platform for text input. In *MobiSys* (2015).

[41] PATHAK, A., HU, Y. C., AND ZHANG, M. Where is the energy spent inside my app?: fine grained energy accounting on smartphones with Eprof. In *EuroSys* (2012).

[42] PATHAK, A., HU, Y. C., ZHANG, M., BAHL, P., AND WANG, Y.-M. Fine-grained power modeling for smartphones using system call tracing. In *EuroSys* (2011).

[43] PLAUMANN, K., MULLER, M., AND RUKZIO, E. Circularselection: Optimizing list selection for smartwatches. In *ISWC* (2016).

[44] POYRAZ, E., AND MEMIK, G. Analyzing power consumption and characterizing user activities on smartwatches: Summary. In *IEEE International Symposium on Workload Characterization (IISWC)* (2016).

[45] QIAN, F., GERBER, A., MAO, Z. M., SEN, S., SPATSCHECK, O., AND WILLINGER, W. TCP Revisited: A Fresh Look at TCP in the Wild. In *IMC* (2009).

[46] QIAN, F., QUAH, K. S., HUANG, J., ERMAN, J., GERBER, A., MAO, Z. M., SEN, S., AND SPATSCHECK, O. Web Caching on Smartphones: Ideal vs. Reality. In *Mobisys* (2012).

[47] QIAN, F., WANG, Z., GERBER, A., MAO, Z. M., SEN, S., AND SPATSCHECK, O. Profiling Resource Usage for Mobile Applications: a Cross-layer Approach. In *Mobisys* (2011).

[48] RAHMATI, A., AND ZHONG, L. Studying smartphone usage: Lessons from a four-month field study. *IEEE Transactions on Mobile Computing 12*, 7 (2013).

[49] ROSEN, S., NIKRAVESH, A., GUO, Y., MAO, Z. M., QIAN, F., AND SEN, S. Revisiting Network Energy Efficiency of Mobile Apps: Performance in the Wild. In *Proc. ACM IMC* (2015).

[50] SANTAGATI, G. E., AND MELODIA, T. U-Wear: Software-Defined Ultrasonic Networking for Wearable Devices. In *MobiSys* (2015).

[51] SHEN, S., WANG, H., AND CHOUDHURY, R. R. I am a smartwatch and i can track my user's arm. In *MobiSys* (2016).

[52] SHEPARD, C., RAHMATI, A., TOSSELL, C., ZHONG, L., AND KORTUM, P. LiveLab: Measuring Wireless Networks and Smartphone Users in the Field. In *HotMetrics* (2010).

[53] WANG, H., LAI, T. T., AND CHOUDHURY, R. R. Mole: Motion leaks through smartwatch sensors. In *MobiCom* (2015).

[54] XU, Q., ERMAN, J., GERBER, A., MAO, Z. M., PANG, J., AND VENKATARAMAN, S. Identifying Diverse Usage Behaviors of Smartphone Apps. In *IMC* (2011).

[55] ZHANG, L., TIWANA, B., QIAN, Z., WANG, Z., DICK, R., MAO, Z. M., AND YANG, L. Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones. In *CODES+ISSS* (2010).

[56] ZHANG, Y., BRESLAU, L., PAXSON, V., AND SHENKER, S. On the Characteristics and Origins of Internet Flow Rates. In *Proc. ACM SIGCOMM* (2002).