WearDrive: Fast and Energy-Efficient Storage for Wearables

Jian Huang[†], Anirudh Badam, Ranveer Chandra and Edmund B. Nightingale

[†]Georgia Institute of Technology Microsoft Research

Abstract

Size and weight constraints on wearables limit their battery capacity and restrict them from providing rich functionality. The need for durable and secure storage for personal data further compounds this problem as these features incur energy-intensive operations. This paper presents WearDrive, a fast storage system for wearables based on battery-backed RAM and an efficient means to offload energy intensive tasks to the phone. WearDrive leverages low-power network connectivity available on wearables to trade the phone's battery for the wearable's by performing large and energy-intensive tasks on the phone while performing small and energy-efficient tasks locally using battery-backed RAM. WearDrive improves the performance of wearable applications by up to 8.85x and improves battery life up to 3.69x with negligible impact to the phone's battery life.

1 Introduction

The utility of a mobile device has long depended upon the tension between the device's size, weight and its battery lifetime. Smaller, lighter devices tend to be easier to carry. However, battery lifetime is mainly a function of size. A smaller device must therefore contain a smaller battery making energy a precious resource. The need for durable storage further compounds this problem. Slow flash storage wastes energy by keeping the CPU active for longer period of time [26, 27, 52], yet the use of a battery dictates that durable storage is vital to a device's utility. Likewise, data encryption is energy-intensive [31], but the sensitive nature of personal information that devices collect dictates using appropriate protection mechanism over a durable medium like flash that can be easily detached from a stolen device to retrieve personal data.

On wearables [43, 13, 5, 35], these trade-offs are magnified. Size matters even more since the device is worn on the body, therefore these devices have a very precious energy reserve. A watch that must be charged after a few hours is not very useful. Likewise, these devices generate precious sensor data (e.g., body sensor readings and location) that must be guaranteed against loss and theft.

In this paper, we explore a new approach to storage on wearable devices that does away with local durable storage while leveraging a nearby phone to protect against data loss and theft in an energy efficient manner. The system, called WearDrive, uses only memory on wearables for storage operations to provide performance and energy improvements. It exploits the battery in mobile devices to provide durability for the data in memory. It leverages low-power network connectivity available on wearables to exploit the capabilities of the phone. New data is asynchronously transmitted to the phone, which ultimately performs the energy-intensive operations of storing data with encryption in its local flash.

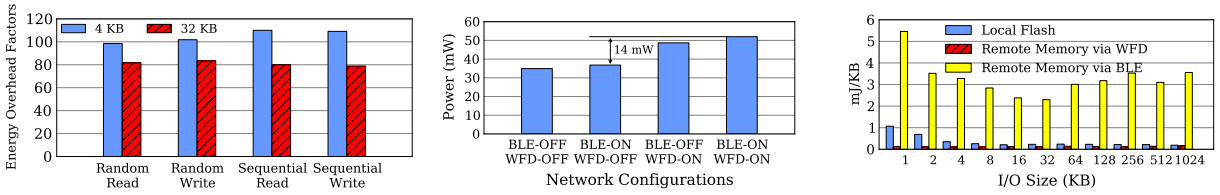
WearDrive targets the two most important application scenarios of wearables. The first scenario is the "extended display" that uses the wearable as a second display to allow applications on a nearby phone to run interactive but less-featured companion applications. Examples include companions that provide notifications for emails, social networks, etc. Providing *fast and durable storage* to such applications helps wearables conserve battery while remaining interactive.

The second scenario is sensor data analysis. Wearables are packed with sensors that take advantage of their location on a person's body. Exposing this data to the applications on the phone with *low-energy data sharing* can open up powerful applications. WearDrive targets these scenarios and reduces the need for a large battery and eliminates the need for flash on wearables. This paper makes the following contributions:

- A distributed battery-backed RAM based storage system called WearDrive is presented that can help applications span data and computation across the wearable and phone quickly and energy efficiently.
- A hybrid Bluetooth and Wi-Fi data transfer scheme is presented. It helps the wearable exploit the capabilities of the phone at a low-energy cost by shipping data and computation to it.
- Data-intensive wearable workloads are identified. A benchmarking tool named WearBench with several data-intensive scenarios is developed to benchmark applications spanning wearable and phone.

Experimental results show WearDrive helps applications obtain up to 8.85x better performance and consume up to 3.69x less energy compared to the state-of-the-art systems with little impact to the phone.

The rest of this paper is organized as follows: Section 2 presents the challenges that we address in this



(a) Software Storage Stack Vs. Storage Hardware (b) Standby Power Consumption (c) Flash Vs. Wireless Network

Figure 1: Motivating scenarios for WearDrive: (a) Mobile storage stacks are energy-intensive because storage software consumes 80–110x more energy than flash. (b) To maintain a connection to the phone for the wearable, WiFi-Direct consumes 10–15mW extra power, while Bluetooth Low-Energy requires only 1–2mW. (c) In terms of energy consumption of the whole system when sequentially writing 32 MB data set with various I/O granularities, it is more energy efficient to write to remote phone's memory via WiFi-Direct than to write data locally to flash on the wearable.

work. Design and implementation of WearDrive are described in Section 3. The benchmarking suite and the evaluation results are shown in Section 4. Section 5 presents the related work. Finally, we describe the conclusions from this work in Section 6.

2 Wearable Storage Challenges

Wearables present a new challenge for mobile system design. Constraints on size and weight limit the battery capacity, but their location on the body and proximity to the phone create new opportunities.

Small Batteries. Li-ion battery metrics like gravimetric energy density (Watt-Hours/kg) and volumetric energy density (Watt-Hours/liter) take 10+ years to double [9]. Therefore, wearables will still be restricted to battery capacities of 1–2 Watt-Hours for the next several years because of their size and weight constraints; today's phones have 7–11 Watt-Hours batteries [19, 44, 33]. Therefore, we propose that *the battery on the phone be traded for the battery on the wearable*.

Energy Overhead of Legacy Platforms. To simplify the hardware and software development of wearables, manufacturers have chosen to reuse the system-on-a-chip (SOC) design and mobile operating systems that were originally made for phones and tablets. For example, most smart-watches and smart-glasses [43, 14, 13, 5] follow this approach to reduce cost, and accelerate development of the platform and the applications. The focus of this paper is on such wearable devices. This means that wearables face a larger energy challenge compared to phones, because of their smaller batteries.

Our prior work [31] identified that mobile storage software consumes up to 110x more energy compared to flash hardware for accessing data as shown in Figure 1(a). The energy overheads are caused by three factors. First, mobile flash is slow and increases CPU idle time while waiting for IO completion [26]. Second, storage on mobile devices is accessed via managed runtime environments like the Darwin engine on Android and the CLR engine on Windows that add additional CPU overhead. Finally, encryption of data that happens using special CPU instructions is also energy intensive. *A fast and energy-efficient storage system with security and privacy guarantees is needed for wearables*.

New Applications. Nearly all existing applications of wearables fall into two categories: extended display and sensor analysis. Using a wearable as an extended display requires arbitrary mobile application state be shared across the wearable and phone. And for wearables, the users focus more on *new content* from contextual applications like email, messaging, social networks, calendar events, music controls, navigation companion and etc.

Wearables are rich sources of sensor data (Table 3) because of their location on the body. For example, watches can better monitor heart-rate and glasses can provide better video sensing. These sensors pave the way for a wide variety of useful applications including long term fitness/wellness tracking, detecting chronic health conditions like sleep-disorder, heart conditions, etc. Existing wearables unfortunately are severely crippled in terms of battery size and provide only limited data analytics. *A storage system capable of supporting these wearable workloads and exploiting their characteristics for performance and energy-savings is needed.*

Reaching the phone. Bluetooth Low Energy (BLE) enables wearables to maintain a constant connection to phone at a low-energy cost (Figure 1(b)). However, its low modulation rate imposes a large energy tax on large data transfers. An alternative is WiFi-Direct (WFD) which requires higher constant power to maintain a connection, but supports low-energy large data transfers with high modulation rates. Figure 1(c) shows the average energy per KB consumed by the whole system of the wearable (see Table 3) as it sequentially writes data to local flash or remotely to the phone via BLE/WFD. The experimental setup is the same as described in Section 4.2. Experimental results indicate that the energy overhead of writing data to remote memory via WFD is comparable to that of writing data to flash on the wearable.

The challenge is to build a mechanism to connect the wearable to the phone with a constant low-power connection overhead with a means to transfer data energy-efficiently. A hybrid connection and data-transfer mechanism can be built using BLE and WFD so that *data sharing between wearable and phone can be enabled at a low-energy cost*.

Slow flash. Mobile flash is slow and energy-intensive [26]. Faster flash technologies like SSDs require 25–100% more $/GB and 5x more energy per operation, and have a controller alone that is bigger than an entire SD card. Moreover, even SSDs are 10,000x slower than DRAM [1]. Furthermore, we demonstrate that data transfers over WiFi-Direct between two mobile devices consumes less energy than writing the same data to flash (Figure 1(c)). We propose that *wearables actively use only DRAM (local and remote) to drastically speed up storage operations*.

3 WearDrive Design

We begin by showing how applications minimize using flash and use mostly DRAM for *fast and durable* storage operations on wearables. We then present a new data management system that helps applications span extended-display and sensor data across the wearable and the phone. A new hybrid BLE/WFD data transfer mechanism is then described which helps WearDrive transmit data at a low-energy cost to the phone.

3.1 Storage with Battery-Backed RAM

To speed up storage operations, WearDrive actively uses DRAM as storage. However, WearDrive guarantees durability in spite of DRAM's volatility. DRAM on mobile platforms is continuously refreshed. The only time when the DRAM refresh stops is when the device is shutdown, the battery runs out of energy or it is removed. The first two scenarios provide an early warning sign allowing data in DRAM to be flushed to flash just in time before the refresh stops. Removing the battery while the system is running can lead to data loss even in today's systems. Moreover, most wearables' batteries are not removable. Therefore, we assume that DRAM can be treated as non-volatile on such devices. We call such DRAM as battery-backed RAM (BB-RAM).

BB-RAM coexists with DRAM to minimize OS changes. It grows and shrinks dynamically according to the memory pressure in the rest of the OS. DRAM is a precious resource on wearable devices. Most of the wearables we surveyed have less than 0.5GB of DRAM. While reserving a known and fixed region of physical memory as BB-RAM simplifies the implementation, it leads to fragmentation of DRAM and does not allow BB-RAM to dynamically expand and contract in accordance with application/OS requirements. WearDrive's BB-RAM design adapts to memory pressure and spans across non-contiguous physical memory pages.

WearDrive uses BB-RAM both on the wearable and phone to ensure high-performance of applications spanning both the wearable and the phone. Wearable uses the phone as the secondary storage for its data. All old data on the wearable's BB-RAM is retired to the phone's BB-RAM. All dirty data in wearable's BB-RAM is also sent to the phone when the wearable needs to shutdown. Likewise, phone uses its flash as the secondary storage for its data in BB-RAM.

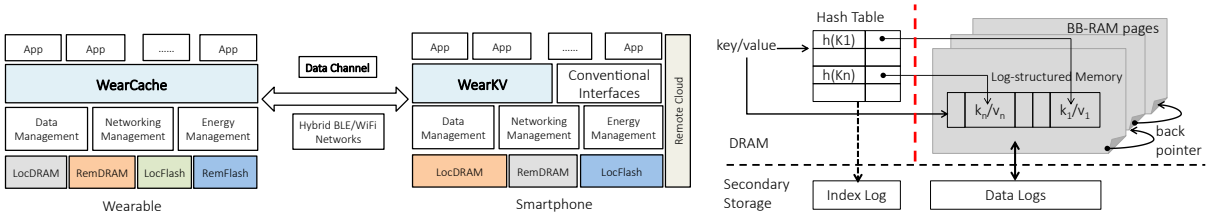
Data in BB-RAM is not lost even after an OS crash. WearDrive uses a firmware component to ensure that BB-RAM is backed to flash in case of an OS crash. Firmware needs additional support to identify the physical pages that are used as BB-RAM. For this purpose, WearDrive reserves a small known region of physical memory to store a bitmap in DRAM to represent whether that physical page belongs to BB-RAM or not. The firmware uses these bits to identify BB-RAM pages after an OS crash (before shutdown) and writes them to a reserved region on flash. This simple design allows BB-RAM to coexist with DRAM and also enables a firmware without any OS state awareness to ensure data durability. Recovering WearDrive's state after a crash solely from the set of BB-RAM pages that it spans across is a harder problem and we present its design in the next sections.

WearDrive uses BB-RAM only as long as there is enough battery life left to ensure durability of data in case of a crash. When battery level reaches a threshold, WearDrive stops using BB-RAM and treats all of DRAM as volatile. New and dirty data is first written to local flash to ensure durability. We set the threshold to 7% in WearDrive based on the observation that flushing 512 MB data from memory to flash sequentially costs about 5% of wearable's battery life on our reference wearable platform. However, this value can be adapted according to the hardware.

Warm reset. WearDrive is optimized for warm resets of the OS. If the available energy is above 7%, the firmware continues to refresh DRAM without scrubbing or cleaning any data. The OS then separates the pages in BB-RAM from regular DRAM using the bitmap and continues the boot process.

OS Deadlock. In case of a deadlock there is a chance that the data in BB-RAM will permanently be lost as the phone is completely drained out of battery. WearDrive uses a watchdog timer to detect if the OS is hung. When the battery life reaches the threshold, firmware schedules a BIOS-context process that wakes up once every sixty seconds and sets a bit in a known portion of memory that it expects the OS to reset every sixty seconds.

[1] Data surveyed from samsung.com, newegg.com and amazon.com



(a) WearDrive Overview

(b) KV-Store Design

Figure 2: (a) WearDrive expands wearable's memory and storage capacity by leveraging phone's capabilities. LocDRAM/RemDRAM represents local/remote DRAM, LocFlash/RemFlash are local/remote Flash. (b) BB-RAM pages are held in a linked list. The pages contain a sequential log of key-value pairs as they arrive. The hashtable stored in regular DRAM contains the index for the key-value store whose state can be efficiently recovered after failures.

If the OS fails to reset it during an iteration then the firmware assumes that the OS has hanged and flushes the data to flash by itself and disables the watchdog timer. The watchdog timer is also disabled as soon as the OS starts using DRAM as volatile.

3.2 Storing Data Across Devices

Since extended display and sensor data analysis scenarios need to span data across wearables and phone, we design WearDrive as a distributed storage system spanning across all devices. We find that in most extended-display scenarios, the wearable is treated as a helper for the full application on the phone because of the smaller screen size on watches, lack of touch screen on glasses and small battery size on both. For this reason, we design the component of WearDrive on the wearable as a cache (WearCache) and the component of WearDrive on the phone (WearKV) as the main storage of data (see Figure 2(a)). WearKV and WearCache both have a key-value store interface that mobile application developers are familiar with. We use the same KV-store system to implement both WearCache and WearKV.

3.3 KV-store Design

KV-store is optimized for BB-RAM. This ensures fast and durable operations for WearCache and WearKV when inserting new data. KV-store prioritizes new data. The focus of wearable applications is on the latest data generated by phone applications and also by the sensors. Examples include the user's interest in latest notifications and most recent sensor values that can provide statistics about a run or a workout session. Therefore, the KV-store is implemented as a sequential log of key-value pairs in BB-RAM with FIFO replacement. Figure 2(b) illustrates the design. Keys and values are arbitrary length data blobs. New values are inserted by appending the KV-pair to the head of the log and adding a hash table entry with pointers to the key and the value in the log.

KV-store stores data in BB-RAM and metadata in DRAM. The log of KV-pairs is stored in BB-RAM and the hash table is stored in regular DRAM. The rationale for this is that the hash table can be recovered from BB-RAM in case of a crash by scanning through the BB-RAM pages in the right order. In case of a clean shutdown, the hash table is serialized to secondary storage (Index Log in Figure 2(b)). This design choice makes effective use of the precious BB-RAM space.

KV-store can recover BB-RAM and DRAM state after a crash. Recall that the firmware flushes BB-RAM pages to local flash in case of a crash. To recover the hash table and the correct head of the log of KV-pairs, ordering of the BB-RAM pages is needed. The ordering of the BB-RAM pages in the log is determined by a four byte pointer stored at the tail of every BB-RAM page to the next BB-RAM page in the log as shown in Figure 2(b). Each KV-pair in BB-RAM is a sequence of five fields: four bytes length of the key followed by the key, followed by eight bytes of application identifier (described later) and then four bytes length of the value followed by the value. This FIFO of BB-RAM pages allows the KV-store to arbitrarily increase its size by appending new pages and decrease the size of the log by purging the KV-pairs at the tail to secondary storage. Moreover, the firmware remains simple, precious BB-RAM space is best utilized and recent data that is of interest for applications is prioritized during page replacement.

WearCache is the KV-store instance that lives on the wearable and caches all the latest data from applications and sensors. New data arrives in WearCache via two methods: when phone applications push data to their companion applications and when sensors generate new values. When WearCache runs out of BB-RAM, it flushes old data to WearKV on the phone in FIFO order as the focus of the wearable is always on new data. It does so by simply moving the tail forward in the log of KV-pairs on BB-RAM several KV-pairs at a time. This provides the functionalities of having recent data on the wearable, adapting to memory pressure, and providing an efficient replacement policy. An example application on today's watches that can leverage this storage model

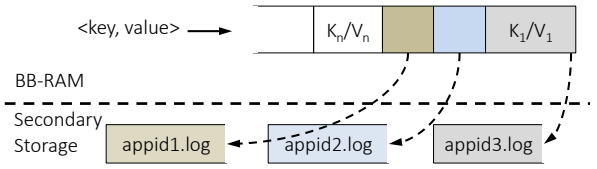


Figure 3: WearDrive creates individual logs per application and per sensor to isolate on secondary storage.

is a notification center for recent emails. The user's focus will be on the most recent emails while the older emails may be safely flushed to WearKV as the user may not access them on the wearable. Complex functionalities are implemented by the email application on the phone while a companion email application on the wearable keeps the design/UI simple with focus on latest data.

WearCache removes flash I/O overhead from the critical path of applications. The OS, application binaries and other application metadata continues to reside on local flash. However, data accessed in critical path resides in WearDrive. The key-value interface to WearDrive eases development as wearable applications already use the key-value interface for sharing data between the phone and wearable [3]. As future work, we wish to provide filesystem and database interfaces using BB-RAM.

WearDrive supports simple sensor data analytics on the wearable and complex data analytics on the phone. Small battery restricts wearables to analyzing sensor logs from short activities like the latest run/workout-session or other short activity. However, applications can perform rigorous analytics on the phone (several days worth of sensor logs at a time). Applications on the phone can proactively pull the sensor data from WearCache as and when a certain number of samples are available. For example, a fitness tracker on the phone can register with WearCache that the heart-rate logs from the wearable be pushed to the phone once every ten minutes. WearCache implements these requests in the following manner. For each sensor, WearCache pre-allocates a KV-pair. A certain amount of space is reserved for the value upfront. The sensor samples (configurable sampling rate) are gradually added to the pre-allocated value as they become available. Data is pushed to the phone and phone-applications are notified accordingly.

WearKV is the KV-store that resides on the phone and contains all the data of the wearable. It contains old extended display data and the entire log of sensor values. Old extended display data is fetched back to WearCache on demand (this is a rare event as wearables focus on new data). The phone with its larger battery can use the full sensor log to perform rigorous sensor data analysis. When WearKV runs out of BB-RAM, it flushes old data to flash where it creates a per-application and per-sensor sequential log as shown in Figure 3. It does so by leveraging the metadata information stored in the values where it records the device-ID, application-ID, sensor-ID and time stamp of creation.

Data in WearDrive crosses the memory/flash boundary only on the phone. Data encryption and other mechanisms put in place to ensure security and privacy of data are needed only for "truly" non-volatile media like flash that can be detached from the rest of the phone and have unprotected data stolen in a straightforward manner. Therefore, the heavy software cost [31] of storage is offloaded to the phone. Note that treating DRAM as non-volatile by using it as BB-RAM is at least as secure as the previous model where data was not encrypted in DRAM as DRAM which is part of the SOC is hard to detach from a device. BB-RAM is a mechanism to ensure that data in DRAM in never lost as opposed to making DRAM "truly" non-volatile.

Offline Capabilities. WearCache can function without the phone. WearCache can lock data on the wearable based on time of arrival such that it is not purged to the phone until explicitly deleted. Offline capabilities allow applications to lock data to be available locally so that functionality can be provided without the phone. An example is when the email companion application imposes a restriction that email from last three days be locked locally. KV-pairs are written to flash on the wearable only if WearCache runs out of BB-RAM and the applications impose an offline availability restriction. Offline requirements are specified in WearCache using time cutoffs per applications and per sensor (see Table 1). We compare the specified time with the timestamp stored in KV-pair's metadata. The qualified offline data is written to its local flash's logs. As time passes, WearCache will move the tail closer to the head on the flash log and overwrites older data that the application does not need.

3.4 Communication

Efficient reachability to the phone allows the wearable to be designed with less DRAM and slower flash thereby reducing their cost. Moreover, it allows the wearable to offload storage and computations to the phone. BLE 4.1 and 802.11a/b/g/n/ac are the network connectivity options for wearables. While a few smart-watches only have BLE, we envision that Wi-Fi will make it to all wearables as it enables efficient large data transfer.

Standalone BLE or WFD is not an ideal network connection. BLE consumes low power (1–3mW) for staying always connected to the phone while using a WFD to stay connected to the phone consumes 5x extra power (10–14mW) (Figure 1(b)). On the other hand, BLE consumes 10–20x extra energy for transmitting data when compared to WFD (Figure 1(c)). A mechanism to minimize the total energy of always staying connected and for transferring data is required.

API	Description
OpenWearDrive (FileName)	open a connection to WearDrive and obtains a handle, the data is represented using an opaque *FileName*.
CloseWearDrive (handle)	close the connection to WearDrive and flush any data from BB-RAM in the process to an appropriate location.
InsertKV (handle, key, value)	insert the new key/value to the *FileName* corresponding to the handle.
ReadKV (handle, key)	provide the value corresponding to the key in the *FileName* file.
MakeOffline (handle, date)	make all data of this file that arrived after a certain date available on the wearable even when the phone is not reachable. Date is specified relatively to the current time. This function is available only to WearCache.
DeleteOldData (handle, date)	provide a hint to WearDrive that data beyond a certain date can be deleted. Date is an absolute value. This function is available only to WearKV.
RegisterForSensor (DeviceID, SensorID)	register an application for values from the sensor represented by *(DeviceID, SensorID)*.
UnregisterFromSensor (DeviceID, SensorID)	unregister the application from a sensor.
RegisterCallBack (TimeGap, CallBackFunction)	make WearDrive issue the *CallBackFunction* in the context of registering application every *TimeGap* seconds with the newly available sensor values.
Compute (DeviceID1, SensorID1, ..., DeviceIDN, SensorIDN, TimeGap)	a function that does not access any global variables but accesses data in sensor logs that are accessible to the application. It can be executed on both wearable and phone.

Table 1: WearDrive API

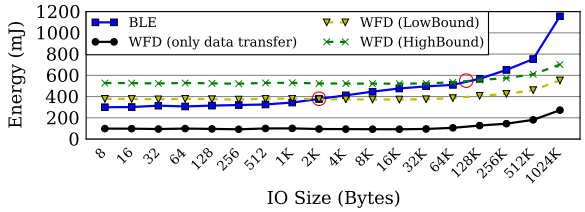


Figure 4: Energy consumption of data transfer via BLE and WFD. WFD is efficient if connection establishment, tail latency and connection-teardown are not included.

Using BLE for staying connected and short data transfers, and turning on WFD solely for large data transfers is a hybrid solution. This is practical because WearCache and WearKV know how much data is to be pushed. If it is beneficial then a control signal over BLE is sent to the other side to turn on WFD. Data transmission begins on BLE and switches over to WFD when it is available.

Knowing the right data transfer size for switching on WFD is crucial. To estimate the transfer size at which it pays-off to turn on the WFD, we conduct the following experiment: transferring data of various sizes on BLE and WFD. We keep BLE always on and send data of various sizes between two mobile devices whose power consumption is monitored using the Monsoon power monitor [36]. We then estimate the energy required for transferring the data via WFD. The energy estimates for WFD contains the energy needed for turning the WiFi chipset on and off. Figure 4 shows the transfer size at which using the hybrid protocol pays off.

The pay-off point for switching to WFD depends on signal quality. We present the results for two extreme modulation rates in 802.11n: the highest modulation rate and the lowest modulation rate. A crossover-point database is built for various modulation rates of BLE and WFD. We use the BLE signal strength to estimate the WiFi signal strength as they use the same band and radio over the same distance.

Picking the right time to turn off WFD is important. WFD consumes more power than BLE in idle state (i.e., standby power gap). However, network discovery, connection and powering-down are expensive, frequently turning WFD on/off would incur more energy usage than keeping it in idle state for workloads with small inter-arrival times. We use two solutions to solve this problem. The first is to have a running average of inter-arrival times and predict on the basis of the average-value if it is worth keeping the WFD on. The second is to explicitly help applications that can tolerate delay to batch data (efficacy evaluated in Section 4) for further energy saving.

3.5 Implementation Details

We implement WearDrive on Android 4.4 using Java, C and JNI [21]. It consists of the KV-store, the data transfer library and the code needed for ensuring durability of BB-RAM. WearDrive is accessed via the calls on *all devices* as shown in Table 1. `InsertKV` and `ReadKV` always append the application ID (stored in `handle`) to the key for inserting and reading data. This helps WearDrive isolate data between applications. Privacy is protected by not providing user-space access to BB-RAM. All data is accessed through user space buffers provided to the system calls.

Sensor values are aggregated by WearDrive on a per-sensor basis. Applications can register sensor logs for each sensor. WearDrive directly appends sensor samples to the pre-allocated KV-pair that is buffering the current set of sensor samples. When enough samples are available, WearDrive notifies the corresponding applications.

4 Evaluation

Evaluating wearable applications is hard because of the lack of a standard benchmarking tool that can generate

Workload	Parameters	Application examples
Extended Display	Size and inter-arrival time distribution of data	Email, news, instant messages, status updates from social networks, etc.
Sensors	sampling rate, monitoring period	Physical fitness, sleep quality, heart health monitoring, elder care, etc.
Audio/ Video	Encoding rate, quality, monitoring period	Dash-cam using glasses, sleep quality monitoring.

Table 2: Workloads included in WearBench.

representative workloads that span across wearables and phone. We present WearBench, a framework that is intended to test the impact of data generated by such wearable workloads on performance and energy.

4.1 WearBench

WearBench is an Android app that runs on the phone/wearable for generating the extended-display data and sensor data which represent wearable applications. WearBench runs on the phone when testing the wearable and vice versa so that WearBench does not interfere with the measurements. WearBench defines synthetic data-analytics that can be executed on sensor logs like calculation of running statistical features including average, standard-deviation, k-means, and hourly/diurnal/weekly pattern recognition algorithms – sampling rate and timeliness are configurable. WearBench can create notifications of varying sizes and different inter arrival time distributions. To the best of our knowledge, WearBench is the first framework for benchmarking wearable systems.

We identify several typical data-intensive workloads running on smart wearables (see Table 2). In order to cover a wide variety of users, we abstract the usage pattern as configurable parameters in WearBench.

The aim of our evaluation is to demonstrate the performance and energy benefits to wearable devices from using WearDrive. We also study the impact on the battery life of the phone. Table 4 summarizes the major benefits of WearDrive for wearable applications over the state-of-the-art methods.

4.2 Experimental Setup

We use a low-end mobile platform as a reference wearable device that runs Android 4.4. As shown in Table 3, our reference wearable device compares to Samsung Galaxy Gear smart-watches which have similar hardware and software configurations. While our reference has 1 GB RAM, we use only 512 MB on it for the system to match the amount of RAM on state-of-the-art wearables.

Monsoon power monitor [36] is used to profile energy consumption of the device. We instrument the reference wearable device's battery-leads such that it draws power from the Monsoon power meter instead of a battery. We

Type	Our Reference Wearable	Samsung Gear
Processor	1.2 GHz dual-core	1.2 GHz dual-core
Memory	1 GB RAM	512 MB RAM
Storage	4 GB eMMC flash	4 GB eMMC flash
Network	Bluetooth 4.0 LE, WiFi 802.11 b/g/n	Bluetooth 4.0 LE, WiFi 802.11 b/g/n
Sensors	accelerometer, barometer, compass, GPS, gyroscope, heart rate monitor, magnetometer, altimeter, barometer, UV light sensor, ambient light sensor, BLE and WiFi events, camera, microphone	accelerometer, gyroscope, compass, heart rate monitor, ambient light, UV light, barometer, GPS, microphone, BLE and WiFi events
OS	Android 4.4	Android 4.3+/Tizen

Table 3: Reference wearable device used for evaluation.

Typical Workloads	Battery-Life Improvements
Passive heart-rate monitoring (Section 4.4.1)	39%
Passive movement monitoring (Section 4.4.1)	54%
Taking pictures (Section 4.4.2)	16%
Taking pictures in burst mode (Section 4.4.2)	27%
Passive video monitoring (Section 4.4.3)	33%
Passive audio monitoring (Section 4.4.4)	50%
Batched Notifications (Section 4.6)	149%
Unbatched Notifications (Section 4.6)	24%

Table 4: WearDrive's benefits for typical wearable workloads compared to Google WearSDK.

perform comparative energy calculations by subtracting the base power of the system from the power used when a workload is executed. However, when reporting absolute energy required for a workload we include the base power of the system. We compare WearDrive with the following state-of-the-art storage solutions:

WearableOnly: The wearable applications use the capabilities on the wearable for storage. The phone is used only for Internet connection via tethering. All the computation is performed locally and all data is durably written to local flash. This is the way most fitness/health trackers are implemented on today's wearables.

WearSDK: Android Wear SDK released by Google [3] is one way to span data across wearable and phone. However, this SDK uses flash synchronously on either one of the devices to ensure durability. WearSDK provides a data layer for data synchronization between paired wearable and phone via BLE (i.e., WearSDK-BLE). We extend the data layer and make it support WFD (i.e., WearSDK-WFD) and our hybrid network protocol (i.e., WearSDK-HYN).

4.3 Local Memory vs. Local Flash

We first examine the advantages of BB-RAM over local flash with a set of microbenchmarks. We configure

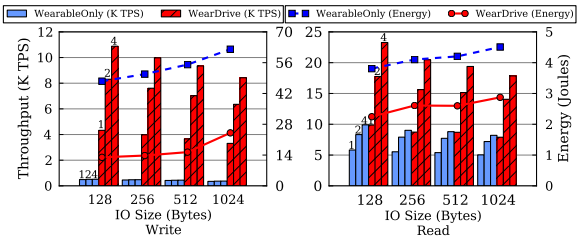


Figure 5: Performance and energy comparison of WearableOnly and WearDrive with varied number (1, 2, 4) of threads.

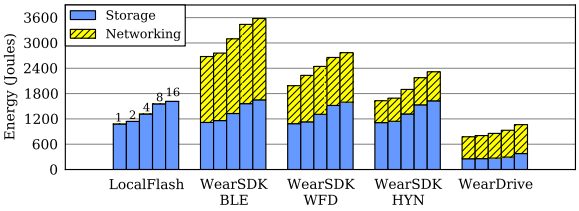


Figure 6: Energy used by various storage systems with varied number (1–16) of sensors sampling values continuously at 1Hz for 24 hours. A typical smart-watch battery contains between 3000–6000 Joules of energy.

WearBench to issue 100 K `InsertKV` and `ReadKV` operations. The size of the data written or read is varied uniformly from 128 bytes to 1 KB. Figure 5 compares the throughput for different data sizes. WearDrive outperforms WearableOnly by 6.65–8.85x on inserts where storage I/O from flash becomes the bottleneck, and 1.57–1.69x on reads where the CPU becomes (single thread) the bottleneck for our system and the flash IOPS for WearableOnly. Moreover, WearDrive's throughput scales linearly till four threads while WearableOnly is saturated by a single thread. Figure 5 also shows the total energy usage of these write/read operations. WearDrive consumes 2.58–3.69x and 1.57–1.70x less power than WearableOnly on inserts and reads respectively, as slow I/O operations on flash cause more CPU cycle wastage, and further increase the energy usage.

4.4 Passive Sensor Data Aggregation

In this section, we demonstrate the benefits of using local and remote BB-RAM for providing durability for sensor data recording over flash.

4.4.1 Fitness Tracker

Fitness/health tracking applications collect sensor values on a periodic basis and update statistics [8]. We use a fitness tracker application that samples various sensors at 1Hz and stores them to local flash periodically. We record the storage calls that this application makes for storing sensor logs, and incorporate the workload into WearBench for replaying.

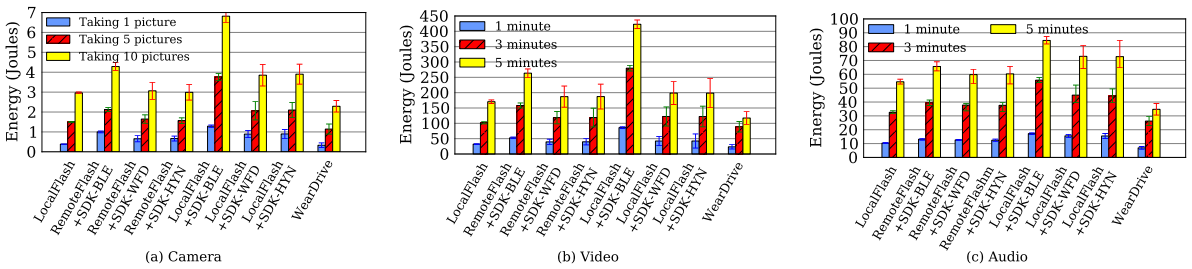
WearDrive aggregates sensor data in BB-RAM and ensures their durability. WearableOnly and WearSDK unfortunately cannot provide such guarantees unless they write every sensor sample through to flash, but they suffer severe performance losses in doing so. As a tradeoff between durability and performance, for these methods, we write the sensor samples to flash when data fills a sector (512 bytes). Every five minutes, all the new data is sent to the phone as sending data to phone at 1Hz leads to significant energy wastage because the network chip would never go into low power mode. Figure 6 shows the total amount of energy in Joules required each day only recording the sensor values. The overall trend across all the systems show that the number of sensors sampled does not severely impact the energy consumption of storage, indicating that the setup costs inside storage stack are the dominant factors for this workload.

WearDrive outperforms the other systems by up to 3.31x and provides better durability. When sampling 16 sensors every second for the whole day and writing them to flash, the storage system (hardware and software) requires 1760 Joules. Considering a typical smart-watch battery that contains 4000 Joules (1.1 Watt-Hour) of energy, writing sensor data to flash requires 44% of total battery life each day. WearDrive on the other hand consumes 28.25%, which is 1.54x more efficient. Moreover, we find that 89.5%, 68.1% and 58.75% of the battery life is respectively required by WearSDK-BLE, WearSDK-WFD and WearSDK-HYN. While HYN reduces the cost of transmitting data over the network to the phone, the bulk of the cost for these systems is still from using slow flash which wastes energy by delaying CPU and network from going to sleep sooner.

4.4.2 Time Lapse Photography

Time lapse photography applications for smart-glasses allow users to log their outdoors activities without the effort of carrying a bulky camera or phone in the hands. We incorporate a time-lapse photography storage workload in WearBench by recording the storage calls of a time-lapse application on Android which takes high-quality pictures at each timer event. Each picture has 2592x1944 dimensions with average size of 900 KB. A few pictures are taken once every few minutes. The results of the workload are shown in Figure 7(a) where average energy required on the wearable per round of photography are reported. LocalFlash stores the pictures only on the wearable. RemoteFlash stores the photos on the phone's flash with the various WearSDK networking solutions. We also test scenarios where photos are stored locally in flash but are also transmitted to the phone with WearSDK. Finally, WearDrive does local BB-RAM to remote BB-RAM copy with HYN.

Results indicate that storing pictures synchronously on



(a) Camera (b) Video (c) Audio

Figure 7: Energy usage on wearable of taking pictures, video recording, audio recording.

local flash is 1.78–2.97x more energy efficient than on remote flash. This implies that eliminating local durable storage is not enough for energy savings and that the energy cost of storage is from the CPU that has to be idle while the flash-IO completes. WearDrive's ability to provide durability for data with local BB-RAM to remote BB-RAM copy reduces the amount of time the CPU and network have to be active and this significantly improves energy efficiency while enabling applications on the phone to have access to the photos.

4.4.3 Passive Video Monitoring

Recording video while traveling, riding or commuting allows a person to have evidence in case of an accident. We add a prototype dash-cam scenario in WearBench that emulates storage calls due to recording of video. Video is shot at 30 FPS (frames per second) with 480p resolution. We buffer video on the wearable for 1, 3 and 5 minutes and then transmit it to phone. Figure 7(b) demonstrates that the energy required scales linearly with data size for large workloads indicating that setup costs in storage show up only for small workloads as in the case of fitness tracking applications.

4.4.4 Passive Audio Monitoring

Some sleep disorders like snoring, bruxism, etc can be diagnosed by passive audio recording on a wearable during the night [53]. We create a prototype passive audio monitoring workload in WearBench by recording all the calls made by an audio recorder application. Audio is sampled for a few minutes continuously several times when the wearable detects motion or noises. The sampling rate for audio is set to 16 KHz, the audio format is PCM 16 bits per sample. As shown in Figure 7(c), compared with the state-of-the-art solutions, WearDrive consumes 1.5x less power than LocalFlash by taking advantages of in-memory store and memory-to-memory data transfer. Combined with the previous results, this provides further evidence that WearDrive can provide benefits regardless of the sensor used as the energy overhead is largely a function of data size.

4.5 Impact on Smart-phone

In this section, we evaluate the energy usage on the phone side and show how WearDrive can improve the lifetime of wearables by leveraging only a negligible portion of phone's larger battery capacity. To understand the energy impact on the phone accurately in this context, we use the same reference hardware in Table 3 as a phone. Note that this is a hardware specification similar to most low-end phones on the market today. However, we use a 2000mAh battery as the reference battery when evaluating the energy impact on the phone.

Energy cost of storage: We reuse the fitness monitoring application workload from Section 4.4.1. Recall that for recording 16 sensors at 1Hz for 24 hours requires 28.25% of the battery life on the wearable instead of 44.0% when writing the data to the flash on the wearable. For this experiment, we find that the phone requires 1369 Joules of energy. This energy accounts for 5.1% of the battery on the phone but this leads to savings of 16% of the battery on the wearable. Considering the fact that the batteries on wearables are usually 5–7x smaller than on low-end phone, this is a valuable tradeoff to make. Moreover, having the data on the phone enables phone to perform analytics and provide more energy savings for the wearable device.

Energy cost of computation: We implement `Mean` and three commonly used data mining algorithms in WearBench: `k-NN` (k-Nearest Neighbor) for classification [24], `ID3` (Iterative Dichotomiser 3) for generating decision tree [20], and `k-means` for cluster analysis [23] for detecting patterns in streams of sensor data to find out when user's heart rate is high [4], when a user snores during the night [53], the levels of UV exposure [51], etc. WearableOnly refers to the baseline, in which records are stored in SQLite and data analytics run on wearables. WearDrive performs computation on the phone with the data in WearKV. The sensor data are aggregated over three days.

Table 5 shows that WearableOnly method of storing and computing on the wearable consumes a significant portion of wearable's battery life, ranging from 14.72% to 27.12%. For smaller data sets the data can be read

Algorithms	Mean		k-NN		ID3		k-means	
Schemes	% of battery life on		% of battery life on		% of battery life on		% of battery life on	
	wearable	phone	wearable	phone	wearable	phone	wearable	phone
WearableOnly	14.72%	-	18.85%	-	20.24%	-	27.12%	-
WearableOnly+InMem	0.83%	-	4.96%	-	6.56%	-	13.23%	-
WearDrive	0.87%	0.21%	0.87%	0.83%	0.87%	1.08%	0.87%	2.09%

Table 5: WearDrive saves wearable's battery by trading it with the phone's battery.The battery capacities of the wearable and phone used in the experiments are 300 mAh and 2000 mAh respectively.

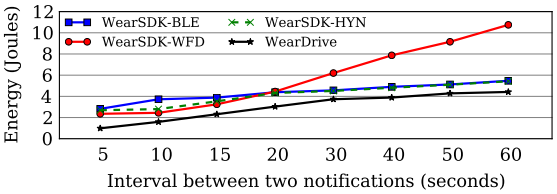
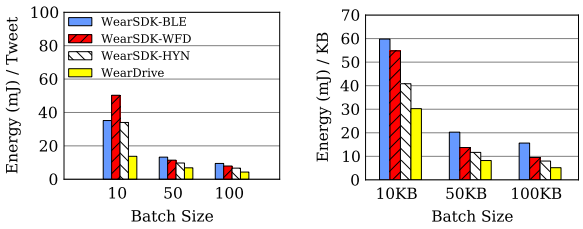


Figure 8: Energy usage of receiving 10 notifications (10KB size) with varied interval between notifications.



(a) Batching Tweets (b) Batching Email

Figure 9: Performance and energy usage of notification workload with different data size.

into memory all at once and computed over as opposed to reading data from flash in batches. We refer to this solution as WearableOnly+InMem. It reduces the energy usage dramatically, but it works only for small workloads that fit in memory. However, when sampled at a higher rate (required usually when the user is running or biking) of over 10Hz, sensor data beyond a few hours will not fit in the memory of the wearable. While such workloads may not fit in the phone's memory either, the phone's larger battery takes much smaller impact.

When the computation is shifted to the phone by WearDrive, it consumes a trivial portion (0.21%–2.09%) of phone's battery life, but reduces the energy usage on wearables to be only 0.87% of wearable's battery life for issuing the arithmetic functions. As future work, we wish to explore when offloading computation to the cloud pays-off with respect to energy. Offloading to the cloud incurs more energy overhead due to data transmission across a wide area with WiFi or LTE. For instance, uploading 8 MB data to Google Drive [11] consumes 3.14x more power than writing to local flash in our experiment setup (with perfect WiFi conditions).

4.6 Extended Display Workload

In this experiment, we demonstrate the benefits of WearDrive to efficiently store extended-display data durably. We use WearBench to emulate application patterns from representative workloads of Twitter [28], Instagram [17], and email [52] applications with various parameters (size and interarrival time).

Varying inter-arrival times. In order to model more notification workload patterns, we vary the interval between tweets from 5 to 60 seconds and measure the energy-impact from storing them durably on the wearable. Figure 8 shows these results.

WearDrive reduces energy usage by 1.2–2.9x compared with the default option of WearSDK-BLE for today's wearable applications. The benefits are made possible not only because of the performance benefits of BB-RAM, but also because of the energy-benefits of HYN. Faster storage operations help the CPU and network go back to sleep faster and reduce the energy footprint.

With HYN, WearDrive uses WFD when the average interval between notifications is small enough to warrant keeping WFD active (20 seconds for our hardware). When the interval is further increased, WearDrive will intelligently turn off WFD and use BLE to send notifications. The hybrid networking protocol also brings benefit to WearSDK (see WearSDK-HYN in Figure 8). For long intervals, WearDrive still performs better than WearSDK-BLE, because of its faster storage.

Effects of batching notifications. Buffering data on the phone gives HYN more opportunity to exploit the energy efficiency of the WFD protocol. We vary the size of the notifications pushed by the phone to wearable from 128 bytes to 1KB. The batch size that the data is sent ranges from 10 to 100. This experiment allows us to study the energy-benefits of delaying notifications from applications that are less interactive than instant messages, such as social networking updates and even email in some cases.

Figure 9(a) shows the results for tweets which are short social networking messages that can tolerate delay. Compared to WearSDK-BLE, WearDrive takes 2.93x less time, while saving energy by 2.23x. The overhead of

WearSDK is reduced with WFD and HYN for large number of notifications. For small number of notifications such as 10 notifications, HYN will use BLE instead of WFD for data transfer. The execution time of WearSDK-WFD is less than WearSDK-BLE and WearSDK-HYN, but its energy usage is larger as the overhead on WiFi discovery and connection offsets its benefit on data transfer. WearDrive is 1.81x more energy efficient than WearSDK-HYN because of BB-RAM's fast durability.

Likewise for email, as shown in Figure 9(b), the benefits of HYN when batching when possible are apparent. However, WearDrive is 2.49x more energy efficient than WearSDK-HYN because of the fast durability guarantee provided by BB-RAM. Overall, WearDrive helps extended-display applications not only by making the energy-batching tradeoff straightforward to exploit but also by providing benefits for applications that are interactive by enabling fast durability.

5 Related Work

WearDrive is built upon existing work on mobile storage systems, hybrid wireless networks and data management for Internet of Things (IoT).

Energy-efficient mobile storage: Kim et al. [26] provided the evidence that slow flash technologies such as SD and eMMC are the primary performance bottleneck for several classes of mobile applications. Our previous work [31] studied the energy overhead of mobile storage systems and found that the mobile software stack consumes more power than storage hardware. These findings motivate our work, as these overheads become more prominent on wearables where the battery is more constrained than on phones.

Recent optimizations to mobile storage [27, 22] address some of the performance problems, but flash is still 10,000x slower compared to DRAM. Emerging non-volatile memory (NVM) technologies like PCM [29, 30, 6] are not yet available in the market. Battery-backed RAM [34, 38, 32] is viable because batteries, DRAM and flash are pervasive in mobile systems. Luo et al. [34] proposed QuasiNVRAM that is a dedicated, known, contiguous region of physical memory to provide performance benefits for phone applications that use SQLite on Android. WearDrive's BB-RAM improves upon QuasiNVRAM by dynamically adapting to memory pressure, not losing data during any class of crashes and by exploiting application characteristics to provide energy and performance benefits.

Rio [45], BlueFS [39], EnsemBlue [40] Simba [1], Segank [47], Bayou [49] and PersonalRAID [46] are distributed file system techniques to share personal data efficiently across mobile consumer electronic devices. WearDrive is an energy-efficient storage system for data intensive wearable workloads like extended-display and sensor data analysis where the workload characteristic of focus on the newest data is exploited to provide a quick and energy-efficient mechanism to span data and computation across the wearable and the phone.

Data management for IoT: Time-series databases [18, 15, 50] enable computations over logs of sensor values. WearDrive is designed to provide time-series data from sensors to applications on the phone at a low-energy cost to enable such computations [37, 54, 42]. Android Wear SDK [3] provides a library to share data between wearables and phone via Bluetooth. WearDrive additionally takes energy-efficiency as its primary design consideration, exploits the recency-focused nature of wearable applications and provides a low-energy durable storage and communication mechanism. WearDrive can also provide sensor data to cloud-based fitness APIs [16, 12, 4] on the phone at a low-energy cost.

Energy-efficient hybrid networks: BlueFi [2], TailEnder [7], Turducken [48], WASP [25], CoolSpots [41] and Bluetooth high speed wireless [10] design heterogeneous networks for efficient data transfer. We draw upon these works and present an energy-efficient hybrid data transfer mechanism by exploiting application knowledge. We find that awareness of data transfer size coupled with the technique where BLE connection is used to predict WiFi's quality enables a mechanism to send data efficiently.

6 Conclusion

WearDrive demonstrates that battery-backed RAM (BB-RAM) can provide significant performance and energy benefits for wearable applications. It also shows how Bluetooth and WiFi can be used in combination to provide a low-energy communication link (HYN) between the wearables and the phone. BB-RAM in combination with HYN provides a quick and energy-efficient way for wearable applications to span data across all the devices on the body enabling new functionalities for users. We validate these benefits with various typical wearable applications using a new wearable benchmarking suite that we develop, and show that WearDrive is 1.16-1.55x more energy-efficient compared to existing solutions. WearDrive can leverage phone's capabilities to reduce energy usage of wearables by up to 15.21x, with trivial impact on phone for realistic wearable workloads.

Acknowledgments

We would like to thank Karsten Schwan and Moinuddin K. Qureshi for their valuable feedback on this work. We also thank our shepherd Theodore Ts'o as well as the anonymous reviewers. This research was performed when the lead author was an intern at Microsoft.

References

[1] AGRAWAL, N., ARANYA, A., AND UNGUREANU, C. Mobile data sync in a blink. In *Proc. HotStorage'13* (San Jose, CA, June 2013).

[2] ANANTHANARAYANAN, G., AND STOICA, I. Blue-Fi: Enhancing Wi-Fi Performance using Bluetooth Signals. In *MobiSys'09* (Krakow, Poland, June 2009).

[3] ANDROID WEAR API. https://developer.android.com/design/wear/.

[4] APPLE HEALTHKIT. https://developer.apple.com/healthkit/.

[5] APPLE WATCH. http://www.apple.com/watch/apple-watch/.

[6] BADAM, A. Impact of Persistent Random Access Memory on Software Systems. *IEEE Computer Society* (May 2013).

[7] BALASUBRAMANIAN, N., BALASUBRAMANIAN, A., AND VENKATARAMANI, A. Energy consumption in mobile phones: A measurement study and implications for network applications. In *Proc. IMC'09* (Chicago, Illinois, Nov. 2009).

[8] BARUA, D., KAY, J., AND PARIS, C. Viewing and Controlling Personal Sensor Data: What Do Users Want? *Persuasive* (2013), 15–26.

[9] BATTERY DENSITY TRENDS RESEARCH BY ARGONNE NATIONAL LABORATORY. http://ec.europa.eu/dgs/jrc/downloads/events/20130926-eco-industries/20130926-eco-industries-miller.pdf.

[10] BLUETOOTH HIGH SPEED WIRELESS TECHNOLOGY. https://www.bluetooth.org/en-us/marketing/high-speed-technology.

[11] GOOGLE DRIVE. http://drive.google.com.

[12] GOOGLE FIT SDK. https://developers.google.com/fit/.

[13] GOOGLE GLASS HARDWARE. https://support.google.com/glass/answer/3064128?hl=en.

[14] GOOGLE GLASS SOFTWARE. https://developers.google.com/glass/tools-downloads/system.

[15] GUPTA, T., SINGH, R. P., PHANISHAYEE, A., JUNG, J., AND MAHAJAN, R. Bolt: Data management for connected homes. In *Proc. NSDI'14* (Seattle, WA, 2014).

[16] HA, K., CHEN, Z., HU, W., RICHTER, W., PILLAI, P., AND SATYANARAYANAN, M. Towards Wearable Cognitive Assistance. In *Proc. 12th ACM MobiSys* (Bretton Woods, NH, June 2014).

[17] HOCHMAN, N., AND SCHWARTZ, R. Visualizing Instagram: Tracing Cultural Visual Rhythms. In *Proc. Sixth International AAAI Conference on Weblogs and Social Media* (June 2012).

[18] HUANG, S., CHEN, Y., CHEN, X., LIU, K., XU, X., WANG, C., BROWN, K., AND HALILOVIC, I. The next generation operational data historian for iot based on informix. In *Proc. SIGMOD'14* (Snowbird, Utah, June 2014).

[19] IPHONE TECHNICAL SPECIFICATIONS. http://www.gsmarena.com/apple_iphone_6_plus-6665.php.

[20] ITERATIVE DICHOTOMISER 3 ALGORITHM. http://en.wikipedia.org/wiki/ID3_algorithm.

[21] JAVA NATIVE INTERFACE. http://developer.android.com/training/articles/perf-jni.html.

[22] JEONG, S., LEE, K., LEE, S., SON, S., AND WON, Y. I/O Stack Optimization for Smartphones. In *Proc. USENIX ATC'13* (San Jose, CA, June 2013).

[23] K-MEANS CLUSTERING ALGORITHM. http://en.wikipedia.org/wiki/K-means_clustering.

[24] K-NEAREST NEIGHBORS ALGORITHM. http://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm.

[25] KAPLAN, M., ZHENG, C., MONACO, M., KELLER, E., AND SICKER, D. WASP: A Software-Defined Communication Layer for Hybrid Wireless Networks. In *Proc. ANCS'14* (Los Angeles, CA, Oct. 2014).

[26] KIM, H., AGRAWAL, N., AND UNGUREANU, C. Revisiting Storage for Smartphones. In *FAST'12* (San Jose, CA, Feb. 2012).

[27] KIM, W.-H., NAM, B., PARK, D., AND WON, Y. Resolving Journaling of Journal Anomaly in Android IO: Multi-version B-tree with Lazy Split. In *FAST'14* (Santa Clara, CA, Feb. 2014).

[28] KWAK, H., LEE, C., PARK, H., AND MOON, S. What is Twitter, a Social Network or a News Media? In *Proc. WWW'10* (Raleigh, NC, Apr. 2010).

[29] LEE, E., BAHN, H., AND NOH, S. H. Unioning of the Buffer Cache and Journaling Layers with Non-volatile Memory. In *Proc. FAST'13* (San Jose, CA, Feb. 2013).

[30] LEE, K., KAN, J. J., AND KANG, S. H. Unified Embedded Non-Volatile Memory for Emerging Mobile Markets. In *Proc. ISPLED'14* (La Jolla, CA, Aug. 2014).

[31] LI, J., BADAM, A., CHANDRA, R., SWANSON, S., WORTHINGTON, B., AND ZHANG, Q. On the Energy Overhead of Mobile Storage Systems. In *FAST'14* (Santa Clara, CA, Feb. 2014).

[32] LOWELL, D. E., AND CHEN, P. M. Free Transactions with RioVista. In *Proc. 16th ACM SOSP* (Saint-Malô, France, Oct. 1997).

[33] LUMIA 930 TECHNICAL SPECIFICATIONS. http://www.microsoft.com/en/mobile/phone/lumia930/specifications/.

[34] LUO, H., TIAN, L., AND JIANG, H. qNVRAM: quasi Non-Volatile RAM for Low Overhead Persistency Enforcement in Smartphones. In *HotStorage'14* (Philadelphia, PA, June 2014).

[35] MICROSOFT HOLOLENS. http://www.microsoft.com/microsoft-hololens/en-us.

[36] MONSOON POWER MONITOR. http://www.msoon.com/LabEquipment/PowerMonitor/.

[37] NAM, Y., RHO, S., AND LEE, C. Physical Acitvity Recognition using Multiple Sensors Embedded In a Wearable Device. *ACM Transactions on Embedded Computing Systems 12*, 2 (2013).

[38] NARAYANAN, D., AND HODSON, O. Whole-system Persistence with Non-volatile Memories. In *Proc. ACM ASPLOS'12* (London, United Kingdom, Mar. 2012).

[39] NIGHTINGALE, E. B., AND FLINN, J. Energy-efficiency and Storage Flexibility in the Blue File System. In *Proc. OSDI'04* (San Francisco, CA, Dec. 2004).

[40] PEEK, D., AND FLINN, J. EnsemBlue: Integrating Distributed Storage and Consumer Electronics. In *Proc. OSDI'06* (Seattle, WA, Nov. 2006).

[41] PERING, T., AGARWAL, Y., GUPTA, R., AND WANT, R. Coolspots: Reducing the power consumption of wireless mobile devices with multiple radio interfaces. In *Proc. MobiSys'06* (Uppsala, Sweden, June 2006).

[42] SAHNI, H., BEDRI, A., REYES, G., THUKRAL, P., GUO, Z., STARNER, T., AND GHOVANLOO, M. The Tongue and Ear Interface: A Wearable System for Silent Speech Recognition. In *Proc. ISWC'14* (Seattle, WA, Sept. 2014).

[43] SAMSUNG GALAXY GEAR SPECS. http://www.samsung.com/us/mobile/wearable-tech/SM-V7000ZKAXAR.

[44] SAMSUNG PHONE TECHNICAL SPECIFICATIONS. http://www.samsung.com/us/mobile/cell-phones/all-products.

[45] SANI, A. A., BOOS, K., YUN, M. H., AND ZHONG, L. Rio: A System Solution for Sharing I/O Between Mobile Systems. In *Proc. 12th ACM MobiSys* (Bretton Woods, NH, June 2014).

[46] SOBTI, S., GARG, N., ZHANG, C., YU, X., KRISHNAMURTHY, A., AND WANG, R. Y. Personalraid: Mobile storage for distributed and disconnected computers. In *Proc. FAST'02* (Monterey, CA, Jan. 2002).

[47] SOBTI, S., GARG, N., ZHENG, F., LAI, J., SHAO, Y., ZHANG, C., ZISKIND, E., KRISHNAMURTHY, A., AND WANG, R. Y. Segank: A distributed mobile storage system. In *Proc. FAST'04* (San Francisco, CA, Mar. 2004).

[48] SORBER, J., BANERJEE, N., CORNER, M. D., AND ROLLINS, S. Turducken: Hierarchical power management for mobile devices. In *Proc. MobiSys'05* (Seattle, WA, June 2005).

[49] TERRY, D. B., THEIMER, M. M., PETERSEN, K., DEMERS, A. J., SPREITZER, M. J., AND HAUSER, C. H. Managing update conflicts in bayou, a weakly connected replicated storage system. In *Proc. SOSP'95* (Copper Mountain Resort, Colorado, Dec. 1995).

[50] THE SCALABLE TIME SERIES DATABASE. http://opentsdb.net/.

[51] UVEBAND. http://www.uveband.com/.

[52] XU, F., LIU, Y., MOSCIBRODA, T., CHANDRA, R., JIN, L., ZHANG, Y., AND LI, Q. Optimizing Background Email Sync on Smartphones. In *Proc. 11th ACM MobiSys* (Taipei, Taiwan, June 2013).

[53] ZEO MOBILE SLEEP MANAGER. http://www.engadget.com/products/zeo/sleep-manager/mobile/.

[54] ZHANG, M., AND SAWCHUK, A. A. USC-HAD: A Daily Activity Dataset for Ubiquitous Activity Recognition Using Wearable Sensors. In *Proc. UbiComp'12* (Pittsburgh, USA, Sept. 2012).