An Integrated Monitoring System for Smartphones

Christopher Miller
University of Notre Dame
miller.444@nd.edu

Sarah Chasins
Swarthmore College
schasil@swarthmore.edu

Carolyn Farris
University of Portland
farris11@up.edu

Justin Varner
Penn State University
jthev05@gmail.com

Curtis Carmony
Bard College
curtis.carmony@gmail.com

Christian Poellabauer
University of Notre Dame
cpoellab@cse.nd.edu

ABSTRACT

Much work has been done in the area of monitoring on traditional systems, such as servers, workstations and laptops. User and application behavior has also been studied on a wide range of platforms. Recently, smartphones have seen a dramatic increase in availability and adoption. New monitoring tools are needed to handle the unique demands of these mobile devices, such as minimal energy usage and cellular network activity, and the unique opportunities they provide, such as incorporating contextual information. Smartphones include a wide range of sensors which can be used to provide insights about the context of the activities being monitored. Individuals also use their mobile phones in a much different manner than traditional systems, and these differences have not been fully explored. With a better understanding of how these devices are used, and how common usage patterns impact system performance, we can improve upon system and application design. In this paper, we introduce an integrated monitoring framework for mobile phones which incorporates sensor, system and user activity. Our expectation is that integrated monitoring solutions will provide the foundation for various new solutions.

1. INTRODUCTION

The rapid increase in smartphones over the past few years is having a significant impact on the primary computing interface that many individuals use. As highly capable mobile devices become more prevalent, more activities are being pushed to these mobile devices, rather than using more robust systems. Much work has been done to study usage on workstations and laptops [6, 1, 12, 7, 2, 4, 9], but typical usage on smartphones can differ significantly. Smartphones are natural candidates for more context-aware applications, since they typically contain multiple sensors, such as GPS and light sensors, which can provide information about the users environment. Understanding how users interact with these devices can help developers optimize applications and system level components to improve performance and make devices more efficient.

While previous work has focused on system and performance monitoring on traditional devices, and on sensor and user activity on mobile devices, we feel that utilizing an integrated cross-layer monitoring tool which incorporates all information provides an opportunity for novel and interesting discoveries. For example, better understanding of the relationship between user activity and sensor readings will allow for analysis of complex socio-technical interdependencies (e.g., how technology affects human activity and interaction and how technology can be improved to better support next-generation social networks). Another example would be to study the interaction between sensor readings or user activity and operating system activity. This could provide new insights into the performance impacts of common usage patterns and applications, which can be used for the optimization of operating systems or applications. We aim to provide an integrated monitoring tool which incorporates the three layers of activity: sensor, system, and user.

We design our monitoring framework for the Android mobile operating system. There are several advantages to working with this operating system. Firstly, Android is an open operating system, so much of the source code is easily accessible. The openness of the platform has also resulted in a large community of developers. Android is based on a standard Linux kernel, which allows us to incorporate methods utilized in Linux. Lastly, Android is a popular and growing platform. It currently sells around 200,000 devices per day, meaning there is a very large and rapidly growing user-base for this platform. Using a combination of kernel patches and kernel modules, we develop an abstracted interface layer for monitoring. In this paper we introduce NDroid, an integrated monitoring solution which provides a simple API for research and development.

2. RELATED WORK

Significant work has been done in the areas of monitoring and modeling on traditional systems. Tools such as PerfMon [7] and PAPI [6] provide low level system performance monitoring. Ganglia [9] and SuperMon [12] provide monitoring tools for clusters and distributed systems. There has been previous work on modeling user or system behavior [4, 1, 2] as well, which is used to improve system performance or design. Some of these same concepts and methods can be utilized in monitoring smartphones, but smartphones present a greater importance on energy conservation, and provide additional contextual information that can be incorporated into the monitoring. Some early work in modeling user activity on smartphones has been completed by Falaki et al. [8].

Previous work has attempted to extract contextual information about users based on embedded or wearable sensors, such as the Mobile Sensing Platform [5]. Recent work has extended this concept to utilize the sensors available in smartphones, since these are becoming ubiquitous wearable sensing devices. Reddy et al. used smartphones to determine the transportation mode of an individual [14], while the CenceMe project attempted to decipher multiple contextual properties such as user activity and location, using a variety of sensors and learning algorithms [11]. More recent work has also attempted to use sensor data from smartphones and user interaction with applications to model human behavior [3, 13, 10, 11]. These studies can provide interesting discoveries which can aid in future system and application design.

3. MONITORING ON MOBILE PHONES

While system monitoring on standard systems has been extensively researched, little work has been done in the area of monitoring on small mobile devices. Mobile phones provide some unique opportunities and challenges for monitoring. Since mobile phones have limited resources in comparison to standard systems, efficiency is imperative. Mobile phones run on batteries, therefore monitoring services should have minimal impact on power usage, so that lifetime of devices is not diminished. Also, since mobile phones have relatively limited processing power and memory, the computational impact on systems should be minimized. Smartphones offer several unique opportunities as well. Smartphones incorporate various sensors, such as GPS, accelerometers, orientation sensors, and ambient light sensors. These sensors can provide context information about the user and their environment. These aspects of usage could not be as easily integrated with system monitoring on standard systems, and could provide new understandings of how systems are used. Also, smartphones are utilized differently than standard systems. This provides an opportunity to understand unique usage patterns on a new platform.

Early efforts to model smartphone usage have shown that there is a great diversity among how users interact with these devices [8]. These findings indicate that optimization of applications may best be achieved using dynamic methods which depend on the user or usage scenario. Developers could incorporate monitoring services into their applications to obtain information which may be useful for optimization. This approach would not be ideal, however, as many developers would likely be incorporating similar monitoring features into their applications, resulting in inefficiencies in the overall system. This is particularly significant in smartphones, where resources are limited, so duplicated monitoring services can be expensive. It can also be difficult to monitor some system level information, especially if it is hardware dependent. Providing a single, simple to interface, abstracted monitoring layer can greatly improve efficiency of monitoring and make it much easier for developers to incorporate monitoring tools into their applications, which could lead to novel improvements. This monitoring layer could handle requests from all applications, and provide a single interface to monitoring tools, which would eliminate inefficiencies due to duplication of monitoring services. As an example, if three applications needed to monitor the CPU load at frequencies of 10 Hz, 20 Hz, and 100 Hz, respectively, the monitoring layer could service all three of these requests using a single 100 Hz monitoring service.

We developed the NDroid monitoring system to be inclusive of all metrics which may be useful to developers or researchers. The features which are monitored in the system may be considered to fall in three categories: sensors, system, and user activity. Sensors include any metrics provided by readings from available sensors on the phone. This feature list will vary somewhat based on the device, but most smartphones have a fairly standard group of sensors, such as GPS and accelerometers. System features include any metrics pertaining to the available resources of the system, and any activities managed by the operating system or kernel. The system resources would include processing power, battery power, memory capacity, and network capacity. System activity would include read/write operations, network actions such as sending a packet, and processor actions such as context switches. All system features are monitored at the kernel level. This is currently accomplished using a patched kernel and kernel modules, but preferably these features would later be pushed upstream to the main Android kernel so that it will be available to all users. To provide an efficient implementation which is consistent with current kernel practices, monitoring features are implemented utilizing the Linux notifier toolchain, which provides a publish/subscribe method of notifications. This method will also allow the monitoring system to customize what is currently being monitored based on application needs, thereby avoiding unnecessary use of system resources. To provide a complete monitoring solution, NDroid will also incorporate user activity monitoring. This includes monitoring of activities such as application usage and text messaging. Integrating this information with system monitoring and sensor data provides the opportunity for stronger analysis of user activity and system performance. The following three sections provide more detail about the three categories of monitoring features, and the specific metrics which are monitored.

3.1 Sensors

Sensors can be used to determine contextual information about the device and user. This data can be used to infer user activity and environmental conditions. Combining this information with system and user activity monitoring may provide unique insights into how user interact with mobile devices. Smartphones include an increasing number of sensors, and research into how these can be used to infer activity or contextual information has grown over the past few years [14, 10]. As research in this area expands, there will only be an increase in the contextual information that can be derived from sensors. A list of the implemented and planned monitoring features for sensors is shown in Table 1.

Table 1: Sensor monitoring features

Activity	Feature	Description
GPS	gps	GPS sensor reading
Magnetometer	magneto	Magnetometer sensor reading
Accelerometer	accelx accely accelz	Accelerometer sensor reading for x, y, and z axis
Orientation	azimuth pitch roll	Orientation sensor reading for azimuth, pitch, and roll
Proximity	proximity	Proximity sensor reading
Light	light	Ambient light sensor reading

3.2 System

System monitoring encompasses most of what would be considered typical performance and system monitoring on traditional systems. In includes information about system resources and operating system activity. System resources encompass the resources available to a system which directly impact user experience. These resources include the processor, memory, and network resources, as well as vital system resources such as the battery. A list of the implemented and planned monitoring features for system resources is shown in Table 2.

Table 2: System resource monitoring features

Activity	Feature	Description
CPU	cpufreq	Current CPU frequency
CPU	cpuload	CPU load
Memory	memavail	Available memory
Cache	cache	Cache in use
Wifi Network	wifibw	Bandwidth of 802.11 interface
Wifi	wifisig	Signal strength of 802.11 connection
Cellular Network	cellbw	Bandwidth of cellular data interface
Cellular Network	cellsig	Signal strength of cellular connection
Battery	battlevel	Remaining battery level (as percentage)
Battery	battcurr	Active current usage of system (in mA)
Battery	battcap	Capacity of battery

Monitoring system resources can provide valuable information for a number of applications. It may be used to optimize the performance of a system by altering the function of the system based on observed usage or available resources, as is done in a userspace frequency scaling application. Applications can use this information to alter their behavior in order to avoid negatively impacting the system or user experience. System resource monitoring may be used to model the impact of applications on a system, and can be a tool for optimizing applications. It may also be used to model user behavior and activity, to determine methods to optimize the system based on usage.

System activities include any action of interest which may be taken by the system, typically utilizing system resources. When integrated with system resource monitoring, these features can be used to study how activities impact system resource usage. When integrated with user activity monitoring, these features can be used to study how user activity impacts the system. These features may also be used for modeling of applications. It is possible that each application has a fingerprint, which can be determined based on observed system activity. Modeling system activity for different applications could provide useful insight to how the system is being used, and what actions may be expected in the future. This could allow developers and researchers an opportunity to optimize system performance based on expected future needs. This information can also be useful for applications, which may need to know the state of some system components. A list of the implemented and planned monitoring features for system activity is shown in Table 3.

Table 3: System activity monitoring features

Activity	Feature	Description
CPU	context	Context switch on the processor
I/O	memread memwrite	Read/Write internal memory
I/O	sdread sdwrite	Read/Write sdcard or external storage
Interface Up / Down	netup netdown	802.11 interface up/down
Interface Up / Down	cellup celldown	Cellular interface up/down
Interface Up / Down	blueup bluedown	Bluetooth interface up/down
Interface Up / Down	gpsup gpsdown	GPS interface up/down
Devices	blueconn bluedisc	Bluetooth device connect/disconnect
Network	nettrx	802.11 packet transmit
Network	netrecv	802.11 packet receive
Network	celltrx	Cellular network packet transmit
Network	cellrecv	Cellular network packet receive

3.3 User activity

User activities encompass activities observed at the application layer. This information can be very useful for understanding typical usage patterns of smartphones. Mobile phones provide a unique environment for system resource usage, which has not been fully explored. Modeling user behavior and developing a better understanding of how these devices are used can be instrumental to efforts to optimize system performance. When combined with system monitoring and sensor data, this data can also provide information about how applications impact system activity and resource usage. This can be a great tool for analyzing and improving application performance. A list of the implemented and planned monitoring features for user activity is shown in Table 4.

Table 4: User activity monitoring features

Activity	Feature	Description
Application	appopen appclose	Open / Close of application by user
Cellular activity	callsnd callrec	Make / Receive a call on cellular network
Cellular activity	textsnd textrec	Send / Receive a text on cellular network
Email	emailsnd emailrec	Send / Receive an email
Screen	screen	On/Off state of screen

4. MONITORING API

The primary purpose of this monitoring tool is to facilitate development and research for smartphones and mobile devices. To do this, we aim to provide a simple and easy to interface API, which will allow developers and researchers to easily access the desired metrics from the system and still provide a robust tool which allows them to customize the monitoring metrics to their specific needs. To accomplish this, we use a kernel module, which serves as a central manager for the monitoring processes, and a central point of communication for the monitoring system and the developer. The NDroid module will expose an API to the developer which will allow them to request the features they want monitored, and the properties for each monitoring activity. The module will then communicate with other modules, and with the kernel, to initiate only those monitoring services which are needed. This will allow the system to provide a very robust monitoring tool without having any greater impact on the system than is necessary. The NDroid API may be accessed by multiple applications, the module will manage which features are needed by each requester, and provide each only the information that is requested.

There are three primary types of requests that can be made to NDroid: instantaneous reading, register a continuous monitor, and register a notification monitor. The instantaneous reading request is simply a request for the current value of a particular feature that can be monitored. For instance, this can be a request for the current load on the CPU. This a single instance request, for cases when a continuous monitor is not necessary. The format for such a request in the NDroid API is:

```
instant_value(u16 FEATURE, (void *) VALUE)
```

`FEATURE` indicates the desired metric to be measured or read, and `VALUE` is a pointer to the location where the value of the metric should be stored.

For metrics that need to be continuously monitored over time, the developer may register a continuous monitor of a supported feature. They can do so by indicating the feature they want to monitor, as well as the frequency of readings or measurements. They will also indicate a callback function for this monitor. This callback function will be used in a similar fashion as the Linux notifier toolchain. The NDroid module will call the function at the requested frequency, including in the call a single argument which is the value of the requested feature. The developer will need to provide a function which handles this value to process it as desired. This method is used to avoid unnecessary reads/writes to the system or to external storage. This helps keep the monitoring tool lightweight by passing the value directly to the requester(s) rather than writing to the proc directory or to a file. The format for such a request in the NDroid API is:

```
register_monitor(u16 FEATURE, time_t FREQUENCY,
(void *) FUNCTION)
```

The NDroid monitoring system can also be used to continuously monitor features but only issue callbacks when certain conditions are met. These monitors will be referred to as notifiers. The monitoring module will continuously monitor these features at the requested frequency, but will only initiate a call to the callback function when the specified criteria are met. The format for such a request is similar to the monitor request, but with the additional information to specify the notification conditions. These conditions may be absolute qualifiers, such as notify when the feature value rises above a certain threshold, or relative qualifiers, such as notify when the feature value changes by a certain threshold. The supported condition types are shown in Table 5. The format for such a request in the NDroid API is:

```
register_notifier(u16 FEATURE, time_t FREQUENCY,
u16 CONDITION, u16 VALUE, (void *) FUNCTION)
```

Table 5: Supported condition types for registered notifier monitor

Condition	Description
MINTHRESH	Notify when metric falls below a minimum threshold
MAXTHRESH	Notify when metric goes above a maximum threshold
CHANGE	Notify anytime there is a change in metric value
UPTHRESH	Notify when metric rises a specified threshold
DOWNTHRESH	Notify when metric falls a specified threshold
ABSOLTHRESH	Notify when metric rises/falls an absolute threshold

5. FUTURE WORK AND DISCUSSION

We are currently implementing the NDroid monitoring tool, and plan to fully implement all monitoring features described above, as well as any additional features which are found to provide useful information. The API module will be implemented to manage each of these metrics, and to expose the described interface to developers for ease of integration. We will also look into alternate methods of monitoring frequency. Rather than leaving frequency of monitoring decisions to users or developers, an alternate option would seek to provide an optimal frequency, such as the Markov-optimal sensing policy proposed in [15]. Following completion of implementation, we will extensively test all features of the monitoring tool to measure its impact on energy usage and processing latency. It is always important to minimize impact on a system when monitoring, but given the limited resources of smartphones, it is especially critical. The monitoring tool has been designed with this in mind, and should be fully tested to ensure that it meets these expectations. Impact to both battery lifetime and system responsiveness should be negligible to users.

Future work will also need to include an examination of the privacy consequences of this tool. The tool will of course be designed to only capture state information which could be useful to understanding usage, such as capturing a text message send event, without capturing the text in the message. Despite this, the tool will have the ability to generate a detailed log of what applications the user used, at what times, and under what environmental conditions (based on sensor readings). This could be considered a privacy control issue, even though specifics of the application use are not captured. This may not be an issue for research which is conducted with a controlled group of active participants. However, if this tool should become an integrated part of Android to support simple access to system resource data by application developers, then these privacy issues will need to be addressed.

There is still much to be learned about how individuals utilize their smart and multimedia phones. Our current knowledge base is rooted primarily in common usage on traditional systems. Usage patterns on mobile phones may, and likely do, differ significantly than usage on traditional systems. Understanding how these devices are utilized can provide valuable information to optimize both hardware and software design, and improve performance and utility. This monitoring tool will be used to capture data on application usage and typical user activity. We hope to use this data to develop models which will provide a better understanding of how applications impact the system and how users utilize smartphone devices. These models could provide the insight needed to improve system performance, or the performance of applications.

6. ADDITIONAL AUTHORS

Additional authors: Aaron Striegel (University of Notre Dame, email: `striegel@nd.edu`)

7. REFERENCES

[1] L. A. Barroso, K. Gharachorloo, and E. Bugnion. Memory system characterization of commercial workloads. In *25th Annual International Symposium on Computer Architecture*, pages 3–14, 1998.

[2] L. Benini, A. Bogliolo, and G. D. Micheli. A survey of design techniques for system-level dynamic power management. *IEEE Trans. VLSI Syst*, 8(3):299–316, 2000.

[3] F. Bentley and C. Metcalf. The use of mobile social presence. *IEEE Pervasive Computing*, 8(4):35–41, 2009.

[4] S. Charbonnier, C. Garcia-Beltan, C. Cadet, and S. Gentil. Trends extraction and analysis for complex system monitoring and decision support. *Eng. Appl. of AI*, 18(1):21–36, 2005.

[5] T. Choudhury, G. Borriello, S. Consolvo, D. Haehnel, B. Harrison, B. Hemingway, J. Hightower, et al. The mobile sensing platform: An embedded activity recognition system. *IEEE Pervasive Computing*, pages 32–41, 2008.

[6] J. Dongarra, K. London, S. Moore, P. Mucci, and D. Terpstra. Using PAPI for hardware performance monitoring on Linux systems. In *Conference on Linux Clusters: The HPC Revolution*, National Center for Supercomputing Applications (NCSA), University of Illinois, Urbana, IL, June 2001.

[7] R. Enbody, K. Pellini, and W. Moore. Performance monitoring in advanced computer architecture. In *Proceedings of the 1998 workshop on Computer architecture education*, page 17. ACM, 1998.

[8] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan, and D. Estrin. Diversity in smartphone usage. In *MobiSys*, pages 179–194. ACM, 2010.

[9] M. L. Massie, B. N. Chun, and D. E. Culler. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(5-6):817–840, 2004.

[10] E. Miluzzo, C. Cornelius, A. Ramaswamy, T. Choudhury, Z. Liu, and A. T. Campbell. Darwin phones: the evolution of sensing and inference on mobile phones. In S. Banerjee, S. Keshav, and A. Wolman, editors, *MobiSys*, pages 5–20. ACM, 2010.

[11] E. Miluzzo, N. Lane, K. Fodor, R. Peterson, H. Lu, M. Musolesi, S. Eisenman, X. Zheng, and A. Campbell. Sensing meets mobile social networks: the design, implementation and evaluation of the cenceme application. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 337–350. ACM, 2008.

[12] R. G. Minnich. Supermon: High-performance monitoring for Linux clusters. In *Proceedings of the 5th Annual Linux Showcase and Conference*, Nov. 2001.

[13] D. Peebles, H. Lu, N. Lane, T. Choudhury, and A. Campbell. Community-Guided Learning: Exploiting Mobile Sensor Users to Model Human Behavior. 2010.

[14] S. Reddy, J. Burke, D. Estrin, M. Hansen, and M. Srivastava. Determining transportation mode on mobile phones. 2008.

[15] Y. Wang, B. Krishnamachari, Q. Zhao, and M. Annavaram. Markov-optimal sensing policy for user state estimation in mobile devices. In *IPSN*, pages 268–278. ACM, 2010.