A Reference Architecture for Mobile Code Offload in Hostile Environments

Soumya Simanta, Grace A. Lewis, Ed Morris
Carnegie Mellon University
Software Engineering Institute
Pittsburgh, PA USA
{ssimanta, glewis, ejm}@sei.cmu.edu

Kiryong Ha, Mahadev Satyanarayanan
Carnegie Mellon University
School of Computer Science
Pittsburgh, PA USA
{krha, satya}@cs.cmu.edu

***Abstract*— Handheld mobile technology can help disaster relief workers and soldiers in the field with tasks such as speech and image recognition, natural language processing, decision-making, and mission planning. However, these applications are computation-intensive, take a heavy toll on battery power, and often rely on good connectivity to networks, limiting their practical usefulness in a crisis. This paper presents a reference architecture for mobile devices that overcomes these limitations by exploiting cloudlets — VM-based code offload elements that are in single-hop proximity to mobile devices.**

Keywords- reference architecture; mobile architecture; mobile systems; code offload; virtual machines; cloud computing

I. INTRODUCTION

First responders and others operating in crisis and hostile environments increasingly make use of handheld devices to help with tasks such as speech and image recognition, natural language processing, decision-making and mission planning [1][2][3]. However, mobile devices offer less computational power than conventional desktop or server computers, and computation-intensive tasks (e.g., image recognition), take a heavy toll on battery power. In addition, networks in hostile environments are often unreliable and bandwidth is limited and inconsistent [4].

Cyber-foraging addresses the challenges of conserving battery power and limited computing power, but does not address the challenges of unreliable networks and dynamic environments [5][6][7][8][9][10][11][12][13]. However, many cyber-foraging strategies rely on the conventional Internet or on environments that tightly couple handheld applications and servers on which code is offloaded. This paper presents a strategy to overcome the challenges of cyber-foraging for mobile platforms in hostile environments by using *cloudlets* — discoverable, localized, stateless servers running one or more virtual machines (VMs) on which mobile devices can offload expensive computation.

II. CLOUDLETS AS INTERMEDIATE OFFLOAD ELEMENTS

Code offload from mobile devices to cloud environments is the topic of many recent papers [14][15][16][17][18][19][20]. However, an underlying assumption in these approaches and solutions is connectivity to the cloud, which is not always the case in hostile environments.

A high-level architecture for code offload in hostile environments is proposed in [21] and presented in Figure 1. At the heart of this architecture is a large centralized core that could be implemented as an enterprise cloud (e.g., Amazon EC2) located in a stable and secure environment. At the edges of this architecture are offload elements for mobile devices. These elements, or cloudlets, are dispersed and located close to the mobile devices they serve [22]. This architecture decreases latency by using a single-hop network and potentially lowers battery consumption by using WiFi or short-range radio instead of broadband wireless which typically consumes more energy [23][24]. A key attribute of this architecture is that the offload elements are stateless. A mobile device does not need to communicate with the core during an offload operation; it only needs to communicate with its physically closest offload element.

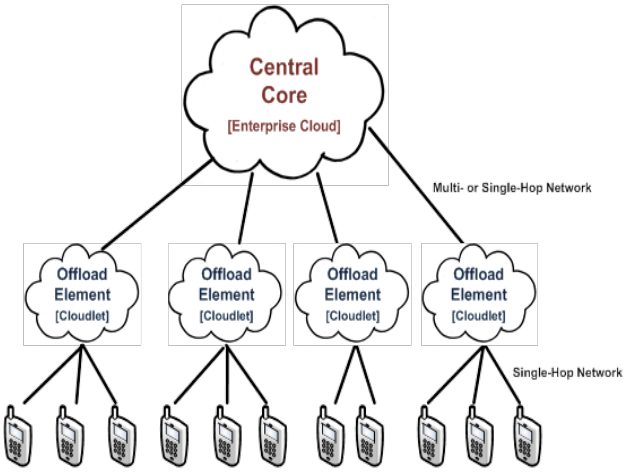

Figure 1. Three-tier architecture for code offload

One approach to offload is virtual machine (VM) synthesis [22][25]. In this approach, an application overlay is offloaded from the mobile device to a cloudlet. An application overlay represents the difference between a base VM with only an operating system installed and a VM with the application installed. As long as the base VM image and associated VM manager are available on the cloudlet, the application can be executed.

An application overlay is created once per application. This process involves obtaining a *Base VM* (a VM disk image file) from the *Central Core* and saving it to a cloudlet that runs a VM manager compatible with the *Base VM*. The VM manager starts the *Base VM*, the application is installed to create a *Complete VM Disk Image*, and the VM is shut down. The *Application Overlay* is calculated as the binary diff (VCDIFF rfc3284) between the *Complete VM Disk Image* and the *Base VM Disk Image*. The *Base VM* is then deployed to physical machines that will serve as a cloudlet. A mobile device carrying application overlays will be able to execute these applications on any cloudlet that has the corresponding *Base VM*.

III. Reference Architecture

A reference architecture for mobile devices exploiting cloudlets for code offload is presented in Figure 2. The major components of this architecture are the *Cloudlet Host* and the *Mobile Client.* The *Cloudlet Host* is a physical server that hosts (1) a *Discovery Service* that broadcasts the cloudlet IP address and port to allow mobile devices to find it, (2) the *Base VM Image* that is used for VM synthesis, (3) a *Cloudlet Server* that handles code offload of application overlays, performs VM synthesis and starts guest VM instances with the resulting VM images, and (4) a *VM Manager* that serves as a host for all guest VM instances that contain the computation-intensive server component of the corresponding mobile app. The *Mobile Client* is a handheld or wearable device that hosts a *Cloudlet Client* app that discovers cloudlets and uploads application overlays to the cloudlet, and a set of *Cloudlet-Ready Apps* that operate as clients of the server code running in the cloudlet. The *Mobile Client* stores an application overlay for each cloudlet-ready app that a user would conceivably want to execute and for which computation offloading is appropriate. Each application overlay is generated from the same *Base VM Image* that resides in the cloudlet. In an operational setting, these *Base VM Images* could be retrieved from the central core shown in Figure 1.

In order to validate the feasibility of the proposed reference architecture for hostile environments, we constructed two prototypes as described in the following sections.

IV. Protoype I

The initial prototype used only disk image application overlays. The following subsections discuss the relevant elements of this prototype.

A. Base VM Image Creation

The Base VM Images for this prototype were standard Windows XP and Ubuntu 11.10 operating systems plus the necessary system updates, with unnecessary system components removed. Additional components were included in each Base VM Image to enable communication between the Guest VM and the Cloudlet Server. After these steps, the VM was shut down and the resulting image file was saved as the Base VM Image.

B. Overlay Creation

The overlay is created using xdelta3 (an open-source binary diff tool [26]), lzma (set of public-domain libraries and tools that compress the diff file using the Lempel-Ziv-Markov chain data compression algorithm [27]), and an open-source SSL (Secure Sockets Layer) implementation that encrypts the compressed file [28].

C. Cloudlet Host

The Cloudlet Host is an Ubuntu 10.10 Linux server that hosts the following sub-components.

1. Kernel-based Virtual Machine (KVM) VM Manager: The virtualization infrastructure used in the prototype is KVM [29]. KVM runs a Guest VM for each offloaded application.
2. Cloudlet Server: The core of the Cloudlet Server is an HTTP Server implemented using CherryPy [30]. The Cloudlet Client sends the overlay to the server using HTTP. Upon receipt of the overlay it is decrypted and decompressed. VM synthesis is performed by using xdelta3 and finally the synthesized VM is started in bridged network mode so that the Guest VM has a unique network-accessible IP address [31]. The Cloudlet Server waits for notification that the Guest VM has successfully started. On startup, the Guest VM communicates the IP address and port back to the Cloudlet Server by using a CloudletStartup Windows Service. The Cloudlet Server is responsible for sending this IP and port address back to the Cloudlet Client.
3. Discovery Service: The Discovery Service is based on Avahi, an implementation of Zero Configuration Networking (ZeroConf) [32][33]. The Discovery Service broadcasts the Cloudlet Server IP address and port.

D. Cloudlet Client

The Cloudlet Client is an Android app that discovers cloudlets, transmits the overlays to the selected cloudlet, and obtains the IP address of the synthesized VM.

E. Prototype Evaluation

The prototype was evaluated using three computation-intensive applications that are representative of capabilities needed by first responders and soldiers. Each application has a client portion which is an Android app and a server portion that corresponds to the computation-intensive offloaded code that runs in the cloudlet.

- OBJECT: Linux C++ application based on the CMU MOPED object recognition libraries [34].
- FACE: Windows XP C++ face recognition application based on the OpenCV image recognition library [35].
- SPEECH: Windows XP Java application based on the CMU Sphinx-4 speech recognition toolkit [36].

In addition, an overlay corresponding to a NULL application (VM simply started and stopped) serves as a baseline for the analysis of transmission overhead and battery consumption. Application data is shown in TABLE I.

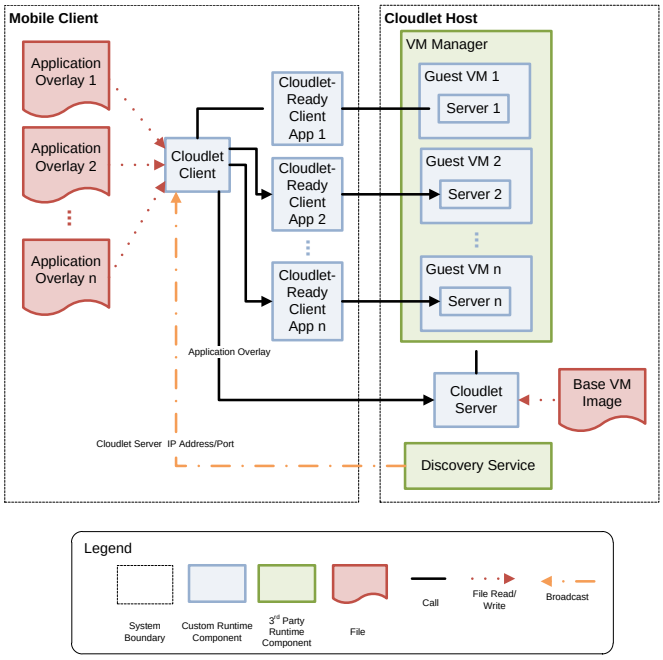

Figure 2. Reference architecture for cloudlet-based code offload

TABLE I. APPLICATION DATA FOR INITIAL PROTOTYPE

Appl.	Platform	Lang.	Appl. Size (MB)	Base VM Disk Image (MB)	Compres. VM Disk Image Overlay (MB)
OBJECT	Linux	C++	27.50	3546	165.32
FACE	Windows XP	C++	17.65	3073	43.55
SPEECH	Linux	Java	51.04	3546	176.23
NULL	Linux	N/A	N/A	3546	0.12

All experiments were conducted on a mobile device running Android 4.0 interacting with an 8-core cloudlet with 72GB of RAM. We measured energy usage with a Monsoon Solutions Power Monitor and the corresponding Power Tool software [37]. To ensure good experimental control, interactive inputs were scripted.

Average times for each step of the process and average energy consumption are shown in Figure 3. All times are measured from the client perspective and include HTTP Request/Response time. The largest amount of time is consumed by the *Upload Overlay* operation and depends on overlay size. The second largest time is almost equally consumed by *VM Synthesis* and *Start VM*. *VM Synthesis* time is smaller for FACE because the sum of the base VM size and overlay size is smaller. FACE is also the outlier for *Start VM* because it is the only application that runs on Windows and therefore has a longer boot time. Energy consumption also depends largely on overlay size. Average application ready time (time between *Upload Overlay* and *Start VM*) is between 101 and 166 seconds for the non-null applications. Given the dependence of these numbers on Base VM Image size and Application Overlay size, reducing the size of these files would reduce both application ready time and energy consumption.

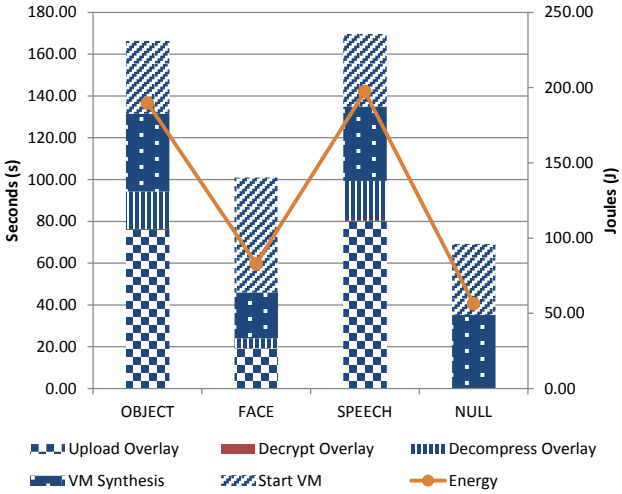

Figure 3. Measures per application for Prototype I

V. PROTOTYPE II

Goals for revising the initial prototype were to reduce application complexity and dependencies and to decrease overall application ready time. Main changes were:

1. Disk Image Format: The default format for KVM disk images is raw images. KVM also supports *QEMU copy on write 2* (qcow2) image format. The advantage of qcow2 is that storage allocation is delayed until it is actually needed thereby optimizing the overlay and image size.
2. Memory Snapshot Overlay Plus Disk Image Overlay: The initial prototype only transfers the disk image overlay. Therefore, the VM is always cold-started and requires application-specific scripts to start the application and send connection information back to

the client. In the revised prototype, the application is installed and launched inside the VM. The VM is suspended, and the memory image is saved along with the disk image and an additional memory overlay is created. In this case total overlay size increases because memory snapshots are large and therefore increase the overall transfer time, bandwidth required and battery power consumed. However, startup scripts inside the VM are eliminated thereby reducing dependencies inside the VM. Additionally, a suspended VM can start faster compared to a cold start.

3. KVM in NAT Mode and Port Redirection: One of the main complexities of the initial prototype was the VM-to-Cloudlet-Host communication. The revised prototype starts the synthesized VM in NAT (Network Address Translation) mode where the Guest VM is not directly accessible from the client. In NAT mode, the Cloudlet Server includes the port that the Guest VM should listen on as a parameter. The Cloudlet Server maps an externally-accessible port on the Cloudlet Host to the port assigned to the Guest VM. The tradeoff is that NAT is restricted to certain protocols. However, this decision greatly simplifies deployment a special startup service inside the VM is no longer necessary.

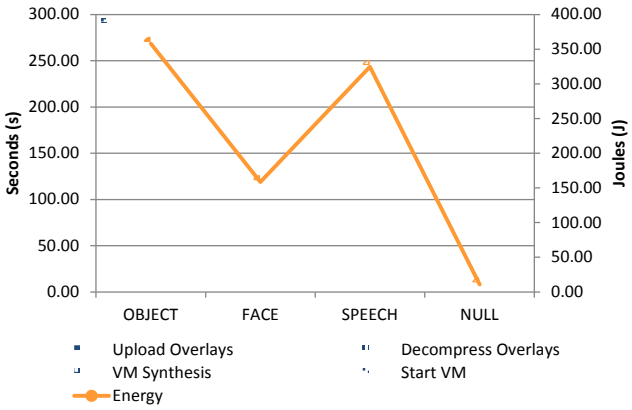

Figure 4. Measures per application for Prototype II

Average times for each step of the process and average energy consumption for Prototype II are shown Figure 4. Although the reduction in *VM Synthesis* and *Start VM* times is considerable with respect to the initial prototype, *Upload Overlay* and *Decompress Overlay* times are higher because of the total size of the combined overlays, also increasing energy consumption. However, implementation, application configuration and VM-to-host-guest communication are simplified, and *Start VM* time is more consistent regardless on the operating system running inside the VM. The bandwidth between the mobile device and the cloudlet during the experiments was approximately 13 Mbps even when using 802.11n wireless. This lower than expected data rate may be caused by radio interference in the environment where the experiments were conducted. The bottom line is that the efficiencies gained in *VM Synthesis* and *Start VM* would have to be supplemented with greater bandwidth for the revised prototype to pay off.

VI. Related Work

There is a considerable amount of work, as early as 2001, related to code offload from mobile devices to cloud environments [5][6][7][8][9][10][11][12][13][14][15][16][17][18][19][20]. However, this work in cyber-foraging from mobile devices assumes that acceptable networking conditions prevail between a mobile device and its offload site. To the best of our knowledge, our work is the first to investigate the challenges of cloud offload in hostile environments and to propose an architectural solution to the problem.

One example of closely-related work is MAUI [15]. MAUI enables fine-grained energy-aware offload of mobile code to offload elements, with minimum programmer effort — code annotations indicate methods that could be executed remotely. However, this approach is platform-specific (Microsoft .NET) and would therefore limit the applications that could be offloaded. Another closely-related effort is CloneCloud [38]. Unlike MAUI, applications do not have to be modified. It also assumes connection to the cloud, but it could potentially be implemented such that threads are migrated to a cloudlet instead of a cloud. However, supported platforms are limited and deployment and hardware requirements would be difficult to achieve in some hostile environments. Other work that establishes a foundation for cyber-foraging includes:

- Goyal et al propose a VM-based approach in which a discoverable virtual machine server acts as a surrogate to run client application code [10].
- The Locusts framework enables the discovery of cyber-foraging resources (surrogate peers). Peers can offload coarse-grained tasks to other peers as well as process offloaded tasks from other peers [39].
- Work by Chen et al is not VM-based but includes a mechanism for deciding whether to execute locally or remotely based on size of method input data and wireless channel conditions, and whether to interpret bytecode or compile native code [40].
- Kemp et al propose work that leverages the Ibis high-performance distributed computing middleware [11]. The server portion is sent from the mobile device to the surrogate at runtime but applications have to be written as distributed applications using the Ibis programming environment and the server.

VII. Conclusions and Future Work

There is substantial consensus that cloud offload of resource-intensive application execution is a core technique in mobile computing. In this paper, we have described a reference architecture for code offload in hostile environments and presented two viable implementations along with architectural tradeoffs. A difficult problem exposed by this architecture is rapid delivery of large application overlays to offload sites and

rapid application ready time. Current and future work includes:

- extension of the discovery protocol to enable VM caching so that overlays do not always have to be transmitted, and by exploiting multi-core architecture to parallelize VM synthesis activities
- mobility-induced cloudlet handoffs to transfer state between cloudlets with minimal interruption to a user
- cloudlet selection mechanism that maps application needs to cloudlet characteristics exposed as cloudlet metadata during the cloudlet discovery process

If cyber-attacks become more prevalent on the public Internet, cloud offload of mobile devices will become increasingly unreliable. Some day, the entire public Internet may have to be viewed as a hostile environment. The issues explored in this paper in the context of first responders will then be of much broader relevance.

REFERENCES

[1] L. Frenzel. Electronic Design: Street-Ready Smart Phone Enhances First Responder Communications: http://electronicdesign.com/article/communications/streetready-smart-phone-enhances-responder-communications-73646 (2012)

[2] J. Kozlowski. EMS1.com: Smartphone and Tablet Apps for First Responders: http://www.ems1.com/ems-products/communications/articles/1129715-Smartphone-and-tablet-Apps-for-first-responders/ (2012)

[3] E. Morris. A New Approach for Handheld Devices in the Military. SEI Blog. http://blog.sei.cmu.edu/post.cfm/a-new-approach-for-handheld-devices-in-the-military (2011)

[4] M. Satyanarayanan. "Fundamental Challenges in Mobile Computing," in Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing (PODC '96). ACM, New York, NY, USA, 1996, pp. 1-7.

[5] R. Balan, J. Flinn, M. Satyanarayanan, S. Sinnamohideen and H. Yang, H. "The Case for Cyber Foraging," in Proceedings of the 10th ACM SIGOPS European Workshop, Saint-Emilion, France, September 2002.

[6] R. Balan, D. Gergle, M. Satyanarayanan and J. Herbsleb. "Simplifying Cyber Foraging for Mobile Devices," in Proceedings of the 5th International Conference on Mobile Systems Applications and Service, San Juan, Puerto Rico, June 2007.

[7] E. De Lara, D.S. Wallach and W. Zwaenepoel. "Puppeteer: Component-based Adaptation for Mobile Computing," in Proceedings of the 3rd Conference on USENIX Symposium on Internet Technologies and Systems, 2001.

[8] J. Flinn, D. Narayanan and M. Satyanarayanan "Self-Tuned Remote Execution for Pervasive Computing," in Proceedings of the 8th IEEE Workshop on Hot Topics in Operating Systems, Schloss Elmau, Germany, May 2001.

[9] J. Flinn, S. Park and M. Satyanarayanan. "Balancing Performance, Energy Conservation and Application Quality in Pervasive Computing," in Proceedings of the 22nd International Conference on Distributed Computing Systems, Vienna, Austria, July 2002.

[10] S. Goyal and J. Carter. "A Lightweight Secure Cyber Foraging Infrastructure for Resource-constrained Devices," in Proceedings of the 6th IEEE Workshop on Mobile Computing Systems and Applications, 2004.

[11] R. Kemp, N. Palmer, T. Kielmann, F. Seinstra, N. Drost, J. Maassen and H. Bal. "eyeDentify: Multimedia Cyber Foraging from a Smartphone," in Proceedings of the 11th IEEE International Symposium on Multimedia, San Diego, CA, December 2009.

[12] M. Ok, J. W. Seo, and M. S. Park. A Distributed Resource Furnishing to Offload Resource-Constrained Devices in Cyber Foraging Toward Pervasive Computing. Network-Based Information Systems, T. Enokido, L. Barolli, and M. Takizawa, Eds., vol. 4658 of Lecture Notes in Computer Science, Springer Berlin/Heidelberg, 2007.

[13] M. Satyanarayanan. Pervasive Computing: Vision and Challenges. IEEE Personal Communications 8, August 2001.

[14] J. Christensen. "Using RESTful Web Services and Cloud Computing to Create Next-Generation Mobile Applications," in Proceedings of the 24th ACM SIGPLAN Conference Companion on Object-Oriented Programming Systems Languages and Applications (OOPSLA '09), 2009.

[15] P. de Leusse, P. Periorellis, P. Watson, and A. Maierhofer. "Secure and Rapid Composition of Infrastructure Services in the Cloud," in Proceedings of the 2nd International Conference on Sensor Technologies and Applications, 2008.

[16] K. Kumar and Y. Lu. Cloud Computing for Mobile Users: Can Offloading Computation Save Energy? IEEE Computer, Vol. 43, pp. 51-56, 2010.

[17] X. Li, H. Zhang, and Y. Zhang. "Deploying Mobile Computation in Cloud Service," in Proceedings of the 1st International Conference on Cloud Computing (CloudCom '09), 2009.

[18] E. Marinelli. Hyrax: Cloud Computing on Mobile Devices using MapReduce. Carnegie Mellon University, CMU-CS-09-164, 2009.

[19] N. Palmer, R. Kemp, T. Kielmann, and H. Bal. "Ibis for Mobility: Solving Challenges of Mobile Computing Using Grid Techniques," in Proceedings of the 10th Workshop on Mobile Computing Systems and Applications, 2009.

[20] X. Zhang, J. Schiffman, S. Gibbs, A. Kunjithapatham and S. Jeong. "Securing Elastic Applications on Mobile Devices for Cloud Computing," in Proceedings of the 2009 ACM Workshop on Cloud Computing Security, 2009.

[21] K. Ha, G. Lewis, S. Simanta, S and M. Satyanarayanan. Code Offload in Hostile Environments. Carnegie Mellon University, CMU-CS-11-146, 2011.

[22] M. Satyanarayanan, P. Bahl, R. Caceres snd N. Davies. The Case for VM-Based Cloudlets in Mobile Computing. IEEE CS Pervasive Computing, 2009.

[23] W. Lehr and L. McKnight. Wireless Internet Access: 3G vs. WiFi? Center for eBusiness @ MIT, 2002.

[24] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra and P. Bahl. "MAUI: Making Smartphones Last Longer with Code Offload," in Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services (MobiSys '10), 2010.

[25] A. Wolbach. Improving the Deployability of Diamond. Carnegie Mellon University, School of Computer Science, 2008.

[26] xdelta.org: xdelta: http://www.xdelta.org (2012)

[27] 7-Zip.org: LZMA SDK: http://www.7-zip.org/sdk.html (2012)

[28] OpenSSL.org: OpenSSL: http://www.openssl.org/ (2012)

[29] KVM: http://www.linux-kvm.org/ (2012)

[30] CherryPy: http://www.cherrypy.org/ (2012)

[31] Ubuntu: KVM/Networking – Community Ubuntu Documentation: https://help.ubuntu.com/community/KVM/Networking (2012)

[32] Avahi.org: Avahi: http://avahi.org/ (2012)

[33] Zeroconf: http://www.zeroconf.org (2012)

[34] MOPED: http://personalrobotics.ri.cmu.edu/projects/moped.php (2012)

[35] OpenCV: http://opencv.willowgarage.com/wiki/ (2012)

[36] Sphinx-4: http://cmusphinx.sourceforge.net/sphinx4/ (2012)

[37] Monsoon Solutions: Power Monitor: http://www.msoon.com/LabEquipment/PowerMonitor/ (2012)

[38] B. Chun, S. Ihm, S, P. Maniatis, M. Naik, M and A. Patti, "CloneCloud: Elastic Execution between Mobile Device and Cloud," in Proceedings of the 6th European Conference on Computer Systems (EuroSys '11), 2011.

[39] M. D. Kristensen, "Execution Plans for Cyber Foraging," in Proceedings of the 1st Workshop on Mobile Middleware (MobMid '08), Leuven, Belgium, 2008.

[40] G. Chen, B. Kang, M. Kandemir, N. Vijaykrishnan, M. Irwin and R. Chandramouli. Studying Energy Trade Offs in Offloading Computation/Compilation in Java-Enabled Mobile Devices. IEEE Transactions on Parallel and Distributed Systems, Vol. 15, No. 9, September 2004.