Can You Help Me Run These Code Segments on Your Mobile Device?†

MINA GUIRGUIS ROBERT OGDEN ZHAOCHEN SONG SOBIT THAPA QIJUN GU
Department of Computer Science
Texas State University-San Marcos
{msg,ro01,zs1044,st1297,qijun}@txstate.edu

***Abstract*— The proliferation of mobile devices, coupled by the increase in their capabilities, have enabled the establishment of a rich mobile computing platform for various applications. In this paper we propose a probabilistic code distribution model that enables a mobile device to execute code segments through the help of nearby mobile devices in a secure and resilient manner. The model relies on randomization and replication techniques against unhelpful devices that do not execute their assigned code segments and malicious ones that try to reveal the overall application. We derive bounds to ensure the success of our scheme with a very high probability. Simulation and implementation experiments using MICAz sensors are conducted to validate our model and study the performance of our scheme.**

I. INTRODUCTION

Over the past few years, we have witnessed a major explosion in the number and capabilities of mobile devices (e.g., cell phones, sensors, robots, etc...). Such devices have grown from simply running a set of limited and specific functionalities to supporting a much larger set of applications. Additionally, most of these devices offer different wireless technologies (e.g., WiFi, bluetooth and cellular) to enable communication to other devices and/or to the infrastructure. The availability of such communication technologies, coupled by the advances in the capabilities of mobile devices, have enabled a rich mobile computing platform for various applications. Recently, there has been some work that aims to harness such mobile computing environment by enabling the partitioning of application between the mobile devices and the infrastructure [1]–[5] to achieve a number of goals, including reducing power consumption, reducing the execution time and ensuring security, among others.

In this paper, we envision applications that require the help of nearby devices in executing code segments. Such applications arise in many scenarios in which executing the whole application on a single device is not feasible due to various constraints such as: (a) limited battery power, since a single device may not have enough battery life to execute the whole application to completion; (b) limited infrastructure access, since a single device may not have access to the infrastructure (e.g., dead zones, remote areas) or there is a high cost incurred (e.g., exceeding the cellular data allowance); (c) different working data sets [6], since nearby mobile devices may operate on local data stored on them (e.g., photos captured within a specific time frame) or acquire data from the environment around them (e.g., collecting sensor information).

Due to the distributed execution of code segments, we must assume the existence of unhelpful and malicious devices. Unhelpful devices do not execute (or return the results of) their assigned code segments. Malicious devices, on the other hand, try to reverse-engineer the application and reveal its functionality/capability. Hence, the integrity, availability and privacy of distributed code execution in mobile networks are inherently threatened by malicious and selfish devices.

Such computation security issues are still unexplored as the majority of existing security research in mobile networks is focused on data and networking security [7]–[9]. Unlike data transmission and management, distributed code execution does not just study how to distribute and manage code, but more importantly how to execute an application over multiple devices. Security on execution is thus different from data-related security issues, and safeguarding execution present new challenges that need new solutions.

In this paper, we propose a probabilistic code distribution model to mitigate the impact of malicious devices that do not contribute to the success of the running application. The idea is to ask additional nearby devices to collectively join their efforts in code execution to obtain accurate results. Therefore, the proposed model utilizes randomness and replication to ensure that the application will execute successfully with a very high probability, even in the presence of unhelpful and malicious devices.

Contributions: The contribution of our work includes: (1) A model which captures the key factors that determine the influence of attackers on the overall success rate of the code execution. In our proposed model, an application is partitioned into a number of code segments. Then, k permutations of the code segments are generated across devices to indicate the assignments of code segments to the devices. We drive the exact number of iterations needed for a high probability success chance. (2) A prototype implementation of the proposed model in a sensor testbed. We conducted various experiments with the prototype to validate the analytical features of our model and evaluate its effectiveness and performance. (3) Our theoretical model is applicable to many other scenarios in which cooperation is envisioned between potentially untrusted

† This work was partially supported by the National Science Foundation under Grant No. 0915318 and the Texas State University One-Time Research Support.

entities (e.g., Peer-to-Peer networks). Moreover, our prototype implementation can be applicable to other devices (e.g., cell phones, robots and other types of sensors).

Paper Organization: In Section II, we present our probabilistic code distribution model and we give a bound on its success probability based on the parameters used. In Section III we present our implementation results using MICAz sensors. We put this work in context with other related work in Section IV and we conclude the paper in Section V.

II. Code Distribution Model

Consider a mobile device that wishes to execute a program P with the help of surrounding mobile devices. We assume that P can be divided into p code segments. We assume the presence of n other mobile devices that may help execute some of the code segments. However, b of those n devices would turn out to be unhelpful, either would not execute or return the results to the requesting mobile device.

The case where $p = 1$ presents three issues; (1) Privacy: the program would be revealed to one device, (2) Power: executing the whole program would likely overwhelm the selected device and (3) No result: if the selected device happens to be one of the b devices, the requesting device would not get any response back. The case where $p = n$ presents the issue that the requesting device would not get the final result since b of the n devices would not respond. Although, there is a potential in finding an optimal p, we assume that the application decides the value of p based on the number of modules that can be partitioned and efficiently distributed to nearby devices.

Assume that the code segments are sent to p distinct devices; some of the segments, m in number say, are sent to helpful devices, and the remaining $p - m$ to unhelpful devices. In effect, we are sampling p devices without replacement from a population of n devices, $n - b$ are helpful and b unhelpful. The classic theory of sampling without replacement tells us that the distribution of the number of segments sent to helpful devices is a hypergeometric distribution:

$$Prob\ [\#\ X_t\ =\ m] \quad = \quad \frac{\binom{n-b}{m}\binom{b}{p-m}}{\binom{n}{p}} \tag{1}$$

where X_t is the set of segments sent to helpful devices on trial t. Clearly, in general there is no guarantee that all the segments will be executed by the selected devices, after one trial of sending p segments to the devices at random. So the requesting device sends the segments out again, and again. Thus, we seek to calculate, and estimate, the number of trials k that the device needs to repeat the above process in order that the helpful devices get the complete program among themselves, with a specified high probability.

Let E_f be the event of failure after k trials. That is, there is at least one segment j=1,...p which never gets executed. Now on a given trial up to k, the probability that segment j is sent to a helpful device is just $\frac{n-b}{n}$. The trials are independent, so the probability that on each of the k trials segment j is sent to an unhelpful device is $(\frac{b}{n})^k$. Now let F_j be the event that segment j is sent to an unhelpful device on each of the k trials. Then the failure event E_f occurs if and only if at least one of the events F_j occurs. Thus we have shown:

$$E_f = \bigcup_{j \in s} F_j, \tag{2}$$

where $s = \{1, ...p\}$ is the set of segments and $|s| = p$. For each $j \in s$ with the $Prob[F_j]$ given by:

$$Prob[F_j] \quad = \quad \left(\frac{b}{n}\right)^k \tag{3}$$

Now the main tool for computing the exact probability of a union of events, such as the expression in Equation 2, is the inclusion-exclusion principle which for this case yields:

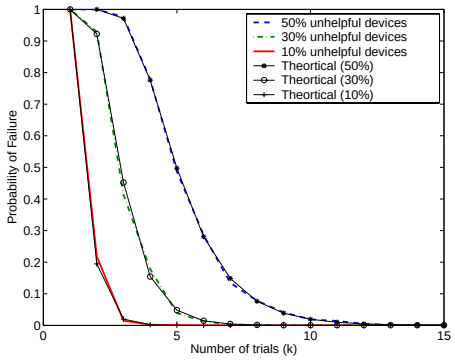
$$Prob\ [E_f] \quad = \quad \sum_{j=1}^{p} (-1)^{j-1} \sum_{q \subseteq s : \#q = j} Prob[\bigcap_{i \in q} F_i] \tag{4}$$

We compute the probability of each event $A_q = \bigcap_{i \in q} F_i$, where the set q has j segments. Now the event A_q occurs if and only if in none of the k trials do any of the segments in q get sent to helpful devices. Since the trials are independent, it suffices to compute, for any one trial, the probability that none of the segments in q get sent to helpful devices. On that trial, say that m of the segments got sent to helpful devices, but that none of these segments are in q. The set of segments sent to a helpful device can be any subset of $s \sim q$, the set of segments not in q, which has m elements. This can be done in $\binom{p-j}{m}$ ways. Since there are $\binom{p}{m}$ subsets with m elements in all, the probability of none of the segments in subset q getting sent to helpful devices is $\frac{\binom{p-j}{m}}{\binom{p}{m}}$, given that the number of segments sent to helpful devices equals m. Thus the probability that no segments in q gets sent to helpful devices on a given trial is:

$$Prob\ [A_q] \quad = \quad \sum_{m=0}^{p} \frac{\binom{p-j}{m}}{\binom{p}{m}} \times \frac{\binom{n-b}{m}\binom{b}{p-m}}{\binom{n}{p}} \tag{5}$$

where the first term in the summation gives the probability that m fragments are sent to helpful devices but none of them is in q and the second term gives the probability of m segments being sent to helpful devices (calculated from Equation 1). The summation is over all possible values of m. Equation 5 can be simplified based on the following:

$$\begin{aligned} Prob\ [A_q] \quad &= \quad \sum_{m=0}^{p} \frac{\binom{p-j}{m}\binom{n-p}{n-b-m}}{\binom{n}{b}} \\ &= \quad \frac{1}{\binom{n}{b}} \sum_{m=0}^{p} \binom{p-j}{m}\binom{n-p}{n-b-m} \\ &= \quad \frac{\binom{n-j}{n-b}}{\binom{n}{b}} = \frac{\binom{b}{j}}{\binom{n}{j}} \end{aligned} \tag{6}$$



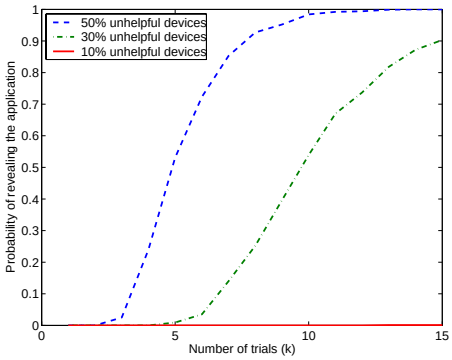


Fig. 1. Theoretical and numerical results. Probability of failure versus the number of serving instants (Left). Probability of revealing the application among colluding devices (Right).

where the last step applies Vandermonde convolution to compute the summation [10]. Now this result holds for any trial, so for k independent trials, we get $Prob[A_q\ after\ k\ trials]$ equals:

$$Prob\ [A_q\ after\ k\ trials] = \left(\frac{\binom{b}{j}}{\binom{n}{j}}\right)^k \quad (7)$$

Since there are $\binom{p}{j}$ subsets of s with j elements, we combine this result with Equation 4 to get the probability of failure:

$$Prob\ [E_f] \quad = \quad \sum_{j=1}^{p}(-1)^{j-1}\binom{p}{j}\left(\frac{\binom{b}{j}}{\binom{n}{j}}\right)^k \quad (8)$$

The formula in Equation 8 is exact, and assuming b is known, the number of trials k required to ensure the probability of failure is less than $\in$ can be computed exactly by the iteration using this formula.

On the other hand, using equation 4 and the fact that the probability that at least one of a list of events occurs is bounded by the sum of the probabilities of the events on the list, we can conclude that $Prob\ [E_f] \leq p(\frac{b}{n})^k$, thus the probability of success after k trials, $Prob\ [E_s]$, is given by:

$$Prob\ [E_s] \quad \geq \quad 1 - p\left(\frac{b}{n}\right)^k \quad (9)$$

Equation 9 may be used to calculate with ease a number of trials sufficient to ensure a stipulated probability of success.

An illustrative example: Consider a set of 10 devices in which 3 are unhelpful ones (denoted by devices 1 through 3) and the remaining 7 are helpful. Consider the case where p is set to 4. Table I shows an example in which it takes 4 iterations ($k = 4$) to get all fragments to the helpful devices.

Figure 1 (Left) shows the probability of failure as the number of trials, k, varies. We do so for 3 different cases of uncooperative devices (50%, 30% and 10%). We plot both, experimental results along with the Equation 5 as a mean to validate the theoretical analysis above. The experiments were based on 1000 runs and in each run, we determine the failure probability. One can see that they match very closely.

	Unhelpful			Helpful						
Device	1	2	3	4	5	6	7	8	9	10
$k=1$		4	1				3	2		
$k=2$		2	4	1			3			
$k=3$	1	3	4			2				
$k=4$			3	1		4			2	

TABLE I

AN ILLUSTRATIVE EXAMPLE FOR OUR METHOD USING 10 DEVICES (THE FIRST 3 ARE UNHELPFUL) WITH $p = 4$ AND $k = 4$.

Probability of not participating: Based on the model above, one can compute the probability of the event that a device will not receive any code segments in the k trials. This probability can be easily proved to be $(\frac{n-p}{n})^k$.

Probability of overlapping segments: Since each iteration is independent from the pervious ones, it is possible that when the permutations are generated, a device may get the same segment. Notice that we do not send this device the segment twice. The probability that a node will receive the same segment k times is given by $(\frac{1}{n})^k$.

Probability of revealing the application by malicious devices: One of our goals is to prevent a set of malicious devices to collude, reverse-engineer the application and reveal its functionality/capability. Figure 1 (Right) shows the probability of revealing the application based on the number of trials. As we increase the number of trials, the malicious devices receive more code segments. We show, however, that this is unlikely to happen for a good choice of k (unless the majority of devices are malicious). We assume the worst case in which all unhelpful devices are colluding. Notice that this can be computed based on Equation (8) (through replacing b with $n - b$). One can see that with 10% unhelpful devices, it is almost impossible to reveal the application (since a good choice of k would be around 5 or 6 from Figure 1 Left). With 30% unhelpful devices, the probability of revealing the application is below 15% for k equals 7 (which achieves a success probability of 0.995 among the helpful devices). With 50% unhelpful devices, the problem becomes similar to computing the success rate among the helpful ones.

III. EXPERIMENTAL EVALUATION

A. Experimental Setup

We implemented a prototype of the proposed code distribution scheme in a testbed made of MICAz sensors. Since the

sensors are typically limited in their computational resources, there is a need to distribute code segments among them when executing a large application to completion.

We implemented an application with multiple segments in the prototype. The completion of the application requires the execution of multiple tasks. Because a MICAz sensor has 128 KByte code memory, the complete application code image with multiple segments is loaded into the sensors. However, we do not ask individual sensors to run all the segments. Instead, we implemented a controller in the sensor network, and let it select sensors and issue commands. Only the selected sensors will execute the segments specified in the commands. Then, the controller will collect the results from the selected sensors and synthesize the final result for the application.

We deployed the prototype in a network composed of 10 MICAz sensors and used 1 base station as a controller. The experimental application has 4 segments. The execution time of the segments is set to 6ms, 9ms, 12ms, and 15ms. Every two seconds, the controller selects 4 sensors randomly to execute the segments. Hence, the duty cycle of each sensor is less than 1%. Among the 10 sensors, we designated b sensors as unhelpful/malicious sensors. In the experiments, we vary b from 1 through 5. To complete the application, the controller selects sensors for task execution over k iterations, where k is set from 2 to 9.

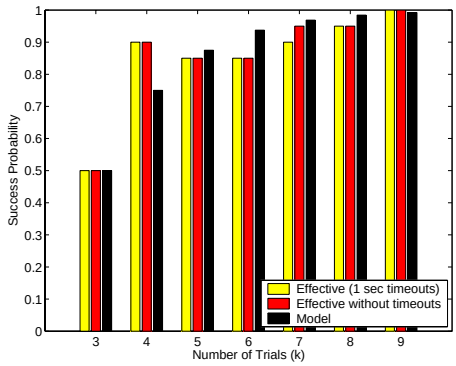


Fig. 2. Implementation results with 10 sensors and 50% unhelpful devices. We show the effect of k on the effective success rate.

B. Model Validation

We have conducted a large number of experiments, under different parameters, to validate our model. Overall we found that the experiments match our model very well. Figure 2 shows a sample of such results. In particular, it compares our model to our experimental implementation with 10 sensors when 50% of them are unhelpful. We show the effect of k on the effective success rate. Each bar is the average result over 20 independent experiments. We present two bars for the experimental part, one with timeouts (the results from good sensors are discarded if they arrive late) and one without timeouts. The timeout value is chosen to be 1 second. One can see that model matches our implementation very well.

C. Overhead Analysis

In the following sets of experiments, we selected 13 pairs of $\{b, k\}$ that can achieve a success rate at 95% or higher according to Equation (9): $\{1, 2\}$, $\{1, 3\}$, $\{2, 3\}$, $\{2, 4\}$, $\{3, 4\}$, $\{3, 5\}$, $\{4, 5\}$, $\{4, 6\}$, $\{4, 7\}$, $\{5, 6\}$, $\{5, 7\}$, $\{5, 8\}$, $\{5, 9\}$. Then, we conducted 20 experiments for each pair of b and k. We measured the communication and the computation overhead of the prototype in the experiments. The communication overhead is measured as the total number of bytes being transmitted in all sensors for issuing commands and collecting results to complete the application. The computation cost is measured as the total CPU time in all sensors for executing the tasks to complete the application. Because malicious sensors are not affected by the overhead, we only measured the overhead incurred by the good sensors.

Figure 3 (Left) and Figure 3 (Center) show the overhead in communication and computation, respectively. To achieve a higher success rate, we would need to increase the number of trials. Consequently, the controller needs to issue more commands and the sensors need to compute the tasks additional times. As illustrated, the increment of both communication and computation overheads is linear to the growth of k.

However, such increment in overhead does not bring much improvement to the success rate, as it is already higher than 95%. In both figures, the solid lines show the minimum overhead to achieve the success rate at 95%. When the number of malicious nodes increases, the minimum overhead to achieve a good success rate does not grow dramatically but rather stabilize. This result indicates that the proposed code distribution scheme can work effectively in a hostile environment with a large portion of attackers.

D. Energy Consumption

In these sets of experiments, we estimate the energy consumption of our proposed scheme. Since the energy consumption is mainly due to communication and computation, it can be modeled as $p_{comm} \times o_{comm} + p_{comp} \times o_{comp}$, where o_{comm} and o_{comp} are the communication and computation overhead that we already measured and p_{comm} and p_{comp} are the power consumption for communication and computation in sensors. According to [11], the MICAz sensor consumes $65.1mW$ when sending and receiving packets and $26.9mW$ in its duty cycle. By plugging in these numbers, we have the estimation of the minimum energy consumption of the prototype, as illustrated in Figure 3 (Right). To complete the application in the experiments, the prototype will consume energy in the range of $2mJ$ and $3.5mJ$ in total as the minimum to achieve the success rate of 95%.

IV. Related Work

One class of mobile computing architectures relies on partitioning application images from smart phones and executing them in traditional cloud services [1], [3]–[5]. In [5], the authors propose an architecture in which the computational infrastructure hosts clones of the mobile devices that can be used in offloading execution. The architecture supports different categories of execution that can be utilized by many application scenarios. In [4], the authors present MAUI to enable fine-grained offload of mobile code to the cloud, in

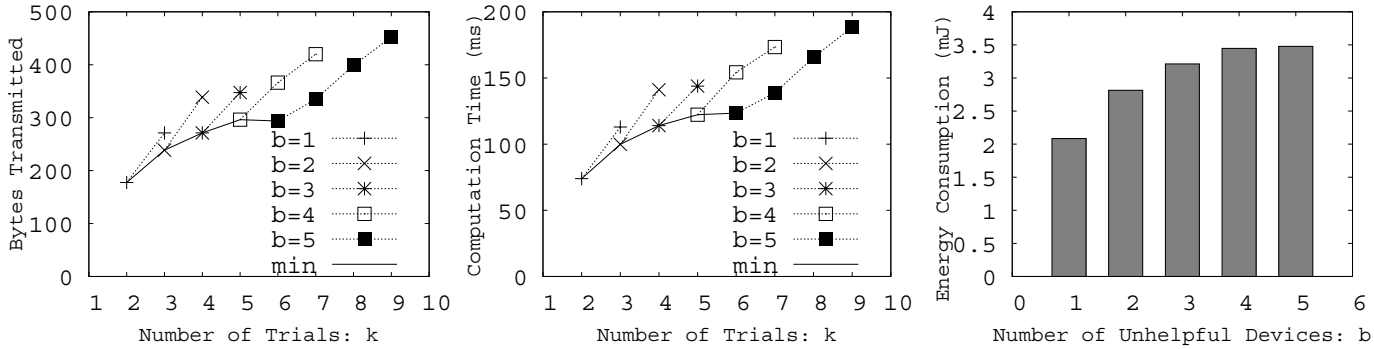


Fig. 3. Left: Impact of k on the communication overhead. Center: Computation overhead among the helpful devices. Right: Estimated energy consumption.

order to save energy on the mobile device. The problem is casted as an optimization one with the goal to maximize the energy saved, subject to meeting the program deadline with a subset of the methods that can be executed remotely. The authors in [3] propose a secure elastic model for applications (called weblets) that can be migrated between devices and the cloud based on the dynamics of the environment. In [1], the authors propose a runtime partitioning method to divide modules between devices and servers on the cloud. The authors formulate the problem as an optimization one (minimizing the execution time or the power consumption) based on the communication and processing costs with a subset of the modules pinned to execute at a specific location. Within this category, some studies advocated executing mobile applications in cloudlets that are one-hop away from the devices [12], [13]. This is done to utilize faster links and reduce transmission delay and energy consumption.

In another class, smart phones are envisioned to share computing capabilities and execute tasks collaboratively in an ad hoc manner among them [14], [15]. In [14], mobile phones form a virtual cloud computing environment to mitigate the constraints on their resources. The authors in [15] propose the RACE framework that is based on a Markovian Decision Process, to enable an optimized bandwidth sharing among smart phones in providing data relays.

In addition to the above studies, some work focused on securing the partitioning of web applications. For example, in [2], the swift system uses a higher-level programming language to explicitly declare security requirement for a compiler to decide on the partitions.

V. Conclusions

Recent advances on mobile devices, in terms of their computing power, battery life and communication capabilities, are opening a rich computing platform that many applications can utilize. In this paper, we have proposed a probabilistic code distribution model in which a device can request nearby devices to execute code segments in a hostile setting. The distribution model identifies the key factors that enable code execution resiliency against unhelpful and malicious devices, and ensures the success rate with a very high probability. In this paper, we have demonstrated our model with a prototype implementation on MICAz sensors.

References

[1] B. Chun and P. Maniatis, "Dynamically Partitioning Applications between Weak Devices and Clouds," in *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*, San Francisco, CA, June 2010.

[2] S. Chong, J. Liu, A. Myers, X. Qi, K. Vikram, L. Zheng, and X. Zheng, "Secure Web Applications via Automatic Partitioning," in *Proceedings of the 21st ACM Symposium on Operating Systems Principles*, Stevenson, WA, October 2007.

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

[4] 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 MobiSys*, San Francisco, CA, June 2010.

[5] B.G. Chun and P. Maniatis, "Augmented Smartphone Applications Through Clone Cloud Execution," in *Proceedings of the 12th Conference on Hot Topics in Operating Systems*, Monte Verita, Switzerland, May 2009.

[6] M. Satyanarayanan, "Mobile Computing: The Next Decade," in *Proceedings of the ACM Workshop on Mobile Cloud Computing*, San Francisco, CA, June 2010.

[7] H. Chan, A. Perrig, and D. Song, "Secure Hierarchical in-network Aggregation in Sensor Networks," in *Proceedings of ACM CCS*, Alexandria, VA, October 2006, pp. 278–287.

[8] M. Shao, Y. Yang, S. Zhu, and G. Cao, "Towards Statistically Strong Source Anonymity for Sensor Networks," in *Proceedings of IEEE Infocom*, Phoenix, AZ, April 2008.

[9] Q. Gu, P. Liu, W. Lee, and C. Chu, "KTR: an Efficient Key Management Scheme for Secure Data Access Control in Wireless Broadcast Services," *IEEE Transactions on Dependable and Secure Computing*, vol. 5, no. 3, pp. 1–14, 2008.

[10] R. Graham, D. Knuth, and O. Patashnik, "Concrete Mathematics: A Foundation of Computer Science," Addison-Wesley Publishing Company, Second Printing, 1988.

[11] Marc Kramer and Alexander Geraldy, "Energy Measurements for MicaZ Node," Technical Report, University of Kaiserslautern, Germany, 2006.

[12] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies, "The Case for VM-Based Cloudlets in Mobile Computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[13] H. Lagar-Cavilla, J. Whitney, A. Scannell, P. Patchin, S. Rumble, E. De Lara, M. Brudno, and M. Satyanarayanan, "SnowFlock: Rapid Virtual Machine Cloning for Cloud Computing," in *Proceedings of ACM European Conference on Computer systems*, 2009, pp. 1–12.

[14] Gonzalo Huerta-Canepa and Dongman Lee, "A Virtual Cloud Computing Provider for Mobile Devices," in *Proceedings of ACM Workshop on Mobile Cloud Computing Services: Social Networks and Beyond*, San Francisco, CA, June 2010.

[15] Eric Jung, Yichuan Wang, Iuri Prilepov, Frank Maker, Xin Liu, and Venkatesh Akella, "User-profile-driven Collaborative Bandwidth Sharing on Mobile Phones," in *Proceedings of ACM Workshop on Mobile Cloud Computing Services: Social Networks and Beyond*, San Francisco, CA, June 2010.