Computing and Informatics, Vol. 30, 2011, 1083–1097

VIRTUALIZING SMARTPHONE APPLICATIONS TO THE CLOUD

Shih-Hao Hung, Jeng-Peng Shieh, Chen-Pang Lee

Department of Computer Science and Information Engineering
National Taiwan University
No. 1, Sec. 4, Roosevelt Rd., Taipei 10617, Taiwan
e-mail: {hungsh, d97026, d97039}@csie.ntu.edu.tw

Abstract. Smartphone technologies have enabled sophisticated pervasive applications for mobile users. Still, many intensive applications perform poorly on smartphones due to the shortage of resources for computation, data storage, network bandwidth, and battery capacity. While such applications can be re-designed with client-server models to benefit from subscribed cloud services, the users are no longer in full control of the entire application execution, which has raised a serious concern. Meanwhile, privacy and security are also important issues, and it is an ongoing debate if public cloud services could be trusted with sensitive data. For mobile users to take full advantage of cloud services, these issues need to be resolved. In this paper, we propose an innovative framework for mobile users to execute existing Android applications on a personal virtual phone safely in the cloud. Instead of using a client-server model, the entire virtual phone is mostly controlled by the user to minimize the intervention from the service provider. Virtualization and encryption are employed to protect against eavesdropping from cloud providers and network attackers. To quickly migrate an Android application between the physical phone and the virtual phone, we use a new application-level checkpointing mechanism and minimize the state of the application.

Mathematics Subject Classification 2000: 68N25

Keywords: Pervasive computing, smartphone, virtualization, cloud applications, information privacy, information security, operating system

1 INTRODUCTION

Smartphone technologies have enabled sophisticated pervasive applications for mobile users. Yet, even the latest smartphones today struggle to perform sophisticated applications as they are constrained by power consumption, speed of computation, size of memory, bandwidth of wireless network, etc. [11]. For a demanding job, it can be beneficial for a smartpohone to *offload* the job onto a server machine using the client-server model to get the job done with less time and save the power consumption on the smartphone [20]. In the age of *cloud-computing*, many cloud-based services are available for processing the workload generated by a smartphone with low amortized operation costs [6]. While the client-server model has been quite successful so far, the following issues have puzzled the users and application developers:

Application re-design and deployment: To adopt the client-server model, the application first needs to be partitioned and a communication protocol is required to connect the client program and the server program. Then, the service has to be deployed on a server and made available to the user, which requires extra efforts and/or costs. For the purpose of offloading the workload in an application, this is often an overkill.

Network condition and service availability: With the client-server model, the quality of the service can depend heavily on the network condition. For a pervasive application, it is impractical to assume that the smartphone phone can always connect to the server with a good network condition. The client-server model would not be able to support the case where the user need to perform the application without any network connection.

Control of the application: When part of the application is processed by a public service provider, the user is no longer in full control of the application. The service provider may use a proprietary interface to trap the users, increase service charges, or go out of business. On the other hand, the user could rent a server in the cloud and deploy the server program just for personal use. However, this would not be an attractive option to average users, in terms of the maintenance effort and the operating costs.

Privacy of personal data: Having personal data processed by a public cloud service also gives the service provider the opportunity to access the data and violate the privacy of the user. In addition to selling personal information to advertisers, social network sites, such as *Facebook* [5], try to promote the sharing of personal information, but it is difficult for the users to manage their privacy setting properly.

Information security: Offloading workload to a public cloud service increases the security risk. The data propagated over Internet and stored in the server could be eavesdropped by the service provider or attackers in the middle. For an average user, it is difficult to know if the service provider is trustworthy or if the server is

secure. For example, an illegal intrusion happened to the PlayStation Network in April 2011, and put the information of 77 million registered accounts at risk.

To address the above issues, we propose a framework for a user to create a *personal virtual phone* in the cloud and allow the user to migrate the execution of an existing application between the user's phone and the virtual phone. The idea is to utilize the personal virtual phone to eliminate the need for creating unnecessary public cloud services to offload computationally intensive smartphone applications. Unlike the client-server model, application redesign is not needed and since the virtual phone is created and controlled by the user, the privacy/security risk is significantly reduced by excluding the service providers.

Since the communication cost for migrating a process [17] or a virtual machine [21] over a mobile network can be prohibitively high, we discuss several strategies to reduce this communication overhead. Then, we further categorize the types of application data to determine on the necessity and the priority of data synchronization. Since the Android platform is relatively new, we have not seen a similar approach in any previous publications. We are not aware of any application-level migration scheme which can migrate existing Android application without re-writing the application code. In addition to offloading workload from the physical phone, the framework gives the user the control of the virtual phone and can work with security measures to protect the information on the virtual phone. It also offers the opportunity to augment the capability of the physical phone, file sharing service, automatic data backup, and virus checking, etc.

The rest of this paper is organized as follows. The proposed framework is described in Section 2. The migration scheme and the communication overhead are evaluated in Section 3. The security of the virtual phone and the implementation of potential value-add features of the virtual phone are covered in Section 4. Section 5 surveys the recent related works and compares them with our work. Section 6 concludes this paper by summarizing our findings.

2 VIRTUAL PHONES IN THE CLOUD

Imagine a personal virtual phone which is connected to high-speed network, has a large storage space, and is capable of performing Android applications several times faster than any physical phone. The owner of the virtual phone may benefit from this virtual phone in many ways. The virtual phone may offload intensive workload from the physical phone to speed up the computation, data access, and network operations. The same application may run on both the virtual phone and the physical phone, so the user may launch the application on either phone or has the application migrated between the two phones. The data of the two phones are synchronized with a cloud storage. As Android applications become the main stream, the user may even use the virtual phone to have server or peer-to-peer applications executed continuously in the background. Since the application is executed

in an environment owned by the user, it greatly reduces the risks of sending sensitive data to a public cloud service for the aforementioned purposes. In this section, we propose a practical framework to facilitate the operation of the virtual phone. We describe the general architecture of the framework and discuss the technical issues.

As illustrated in Figure 1, creation of a virtual phone is automated by our framework as the following procedures.

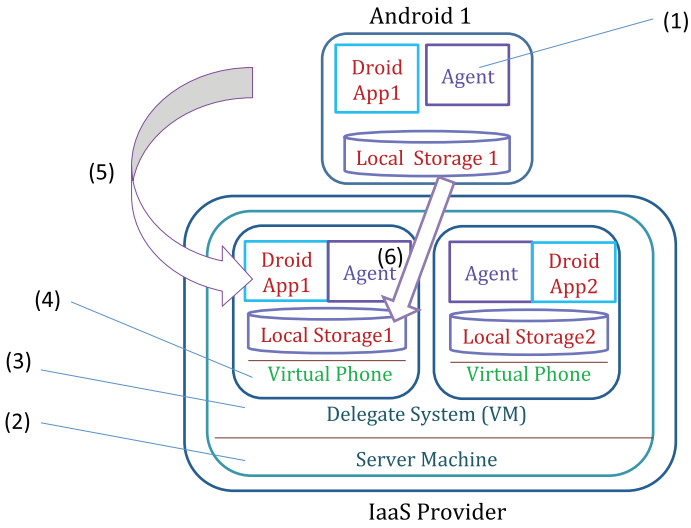

Fig. 1. Procedures for creating a virtual phone in the cloud

1. **Installing our agent program:** The user installs and runs our agent program, which automates the rest of the procedures. The agent also provides the interface for the user and applications to interact with the virtual phone.
2. **Allocation of a delegate system:** The agent allocates a delegate system to host the virtual phone by subscribing to a virtual machine from a trusted infrastructure as a service (IaaS) provider. The delegate system in our prototype runs the Linux operating system and may host multiple virtual phones to save the operation cost.
3. **Setting up a virtual phone:** The agent sets up a virtual phone on the delegate system using the *Google Android Emulator*, an open-source software which employs the *QEMU* [8] virtual machine monitor (VMM) software to emulate the reference Android hardware defined by Google. The QEMU VMM is capable of executing ARM binary codes on an x86-based server with a fast emulation speed, thanks to its a dynamic binary translation scheme. If binary compatibility is not needed by the user, the virtual phone can be set up with Android-x86, which is

much faster than the Android Emulator, but would only support applications written in Dalvik [1].

4. **Cloning of the operating environment:** The agent clones the operating environment of the physical phone or uses a *standard image* stored in the delegate system to create a fresh virtual phone. An exact clone of the operating environment should increase the compatibility between the virtual phone and the physical phone. For applications which require vendor-specific libraries or system services, a full clone is necessary. For other (standard Android) applications, a standard image would be recommended to save the communication costs for cloning the physical phone and accelerate the creation of the virtual phone.
5. **Migration of applications:** The agent on the physical phone takes the commands from the user and relays the commands to the virtual phone. The user may request the agent to migrate a running application from the physical phone to the phone, or vice versa. To migrate an application, the state of the application and the working file set have to be migrated. As the latency for migrating a *live* (executing) application may affect the user experience, it is critical to reduce the latency by minimizing the size of the application state and identifying the working file set.
6. **Synchronization of applications and user data:** As the physical phone and the virtual phone may operate on the same set of files in parallel, the agent programs on both phones need to collaborate to keep the application packages and user data consistent and coherent on both phones. Since continuous synchronizing modified files generate a large amount of network traffics and is often unnecessary, we should pay attention to the policies and protocols for the synchronization of files.

Procedures 1 to 4 are performed once to set up a virtual phone initially. Depending on the need of the user, the actual construction of the framework, and the network condition, the set up time for a virtual phone ranges from a few seconds to a few hours. Setting up a virtual phone with a standard Android OS image to offload portable Android application written in Dalvik is very quick, which makes it practical to locate a nearby server and create a virtual phone on demand.

On the other hand, Procedures 5 and 6 are involved in the daily use of the virtual phone. The latencies caused by these two procedures are important to user's experience, but the issues are more sophisticated. We shall further address the performance issues in the next section.

3 MIGRATION OF ANDROID APPLICATIONS

For migrating a server application, it is commonly done by transferring the *snapshot image* of the virtual machine that runs the application. Unfortunately, this would be impractical to the migration of a mobile application. Given the fact that a snapshot image can easily reach 256 MB for today's entry-level Android phones, transferring

the image over a wireless network should take a long time (e.g. roughly 10 minutes via a 7.2 Mbps HSDPA network). Obviously, it is an overkill instead of migrating the entire system, if what the user wants is to migrate an application.

Process migration is an alternative approach [19]. Unfortunately, process migration is practically too complicated, as it requires an extensive support from the underlying operating system. Even with a working process migration scheme, the working set of an application process can still be large enough to cause a long latency over a mobile network.

The migration scheme in our framework is designed to leverage the application *pause-resume* interface defined by the Android application framework [2]. The application pause-resume interface is specifically designed to support multitasking with limited processing power on an Android phone. When an Android application is switched to the background, it may keep running, but the Android runtime system may opt to close it when the available memory on the phone is running low. In order to have the application resume execution later, the application developer may use the pause interface to have the *application state* saved in the non-volatile memory (e.g. flash memory) and use the resume interface to have the application continue its execution from the previously saved point. This pause-resume interface can be regarded as an application-level checkpointing scheme and used for migrating applications across different Android phones. Since many existing Android applications already follow this paradigm, our migration scheme can migrate them without modifying them. We also encourage application developers to take advantage of the pause-resume interface not only to make their applications more user friendly, but also to enable application migration between Android phones.

The migration scheme in our framework is illustrated step-by-step in Figure 2. On the left-hand side:

1. The agent sends a signal to the application and has the application enter the *OnPause* function.
2. The application saves its state in the *OnPause* function and
3. informs the agent when the state is saved.
4. The agent reads the state and
5. sends the state to the agent on the other side.

Then, on the right-hand side:

6. The agent saves the state and
7. starts the application (or copies the application from the other side if it does not exist).
8. The application resumes by calling the *OnResume* function and
9. resumes the execution after restoring the application state.

Our first case study is a *peer-to-peer* (P2P) file exchange application program, called *androidtorrent* [4]. The purpose is to illustrate the application migration

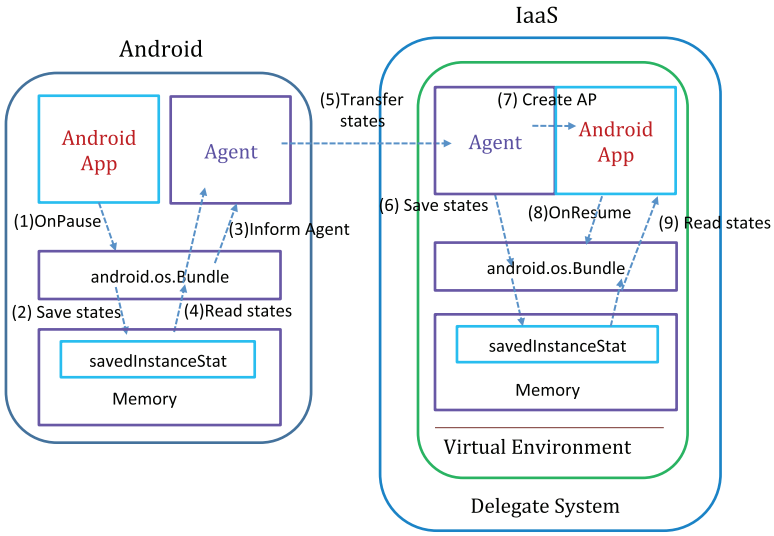

Fig. 2. Illustration of application migration to the cloud

procedure and the capability of our framework. P2P is a distributed network for participants to share their resources without a centralized coordinator [23]. The working set in *androidtorrent* can be quite large when the user exchanges many files with many peers. In practice, while P2P network is popular on personal computers (PC), most smartphone users would not run this application since it is relatively slow to exchange files over a mobile network and the application constantly consumes processor/memory/network/battery resources on the phone.

As we enter the so-called *post-PC era*, like *androidtorrent*, many PC applications face the same challenge on smartphones. While there are *proxy* services in the cloud which would like to replace the P2P client program, as we mentioned in the beginning of this paper, the user would need to subscribe to a service, and the service provider may be aware of the activities and the files processed by the server. So far, we cannot find a proxy program to install the same service on a private server by ourselves. Even if we found one, it would not be an elegant solution for an average user. It is a dilemma: on one hand, it would take some effort and cost for the user to maintain a private server; on the other hand, if a public service is chosen, the privacy of the data may be a concern.

Obviously, this type of workload would be best handled by a virtual phone in the cloud. In our experiment, we created a virtual phone, started *androidtorrent* in a physical phone, selected the files to exchange on the physical phone, and then migrated *androidtorrent* to the virtual phone. During the migration, the agent transferred the states saved by *androidtorrent*, approximately 320 Kbytes of data,

to the virtual phone over a 3G mobile network and resumed execution in 3.8 seconds.

As *androidtorrent* proceeded its execution on the virtual phone, it constantly created new files and modified existing files. If our framework were to enforce continuous synchronization of the filesystems between the physical phone and the virtual phone, a lot of unnecessary network traffics would be generated, since the user has no need for those temporary files. In our framework, we adopt a version of lazy synchronization policy to address this problem. When a file is modified on one phone, the copy on the other phone is invalidated. It is not until the invalidated file is accessed, the file would not be synchronized by the framework. On the other hand, there are system data and files shared with other applications that need to be synchronized immediately, for example contact (addressbook). Since most Android applications are well structured, it is relatively straightforward for the framework to identify the system data and shared files. The agents on both phones monitor and synchronize these files as soon as possible, and work jointly to ensure that no race condition occurs in the synchronization. The details are discussed later in this section.

As shown in Table 1, *androidtorrent* performed significantly better on the virtual phone hosted by a PC in our laboratory, thanks to the faster processor and the fixed ADSL network. The results were shown to prove the concept of our framework. Although the subscribed 3G network offered a 7.2 Mbps peak bandwidth, its actual connection speed depended on many issues and were not as reliable as the ADSL network. The execution of *androidtorrent* consumed a lot of processor power and the memory space on the Google G1 phone and could have easily drained the battery power completely in a couple of hours.

Phone	Processor	Network/Max. Bandwidth	Data Transfer
Google G1	528 MHz ARM	3G HSDPA/7.2 Mbps	36 MB/Hr
Virtual Phone	2.6 GHz Intel Core i7	ADSL/2 Mbps	240 MB/Hr

Table 1. A case study with androidtorrent

In addition to *androidtorrent*, we have studied many other Android applications to further evaluate the cost of migration. We categorized the files associated with an Android application into five types: *program package*, *configuration files*, *database files*, *cache buffers*, and *saved state files when the application is on pause*. Table 2 lists the the total size of the file(s) in each category for ten popular Android applications. To migrate a new application to the virtual phone, the package has to be transferred only once. The configuration files and database files are monitored closely in our framework, so any modifications to these files are synchronized immediately via the shared storage. For example, *Contact Provider* is part of the Android middleware which maintains a database for applications to find contact information, and it is important to synchronize the database files even though Contact Provider is not running on the virtual phone. The cache buffers are primarily used by streaming multimedia application, such as YouTube, to prefetch multimedia contents. Finally,

the state files for an application need to be synchronized only when the application is being suspended and migrated onto another phone.

Applications	Package	Config.	Database	Cache Buffer	State
Sudoku	740 888	146	0	0	183
AndFTP	562 795	0	0	0	1 464
aBTC	92 432	0	0	0	144
YouTube	328 622	0	0	397 622	577
Music Download	438 256	0	0	0	119
Music Player	339 323	0	0	0	247
Google Translate	951 909	0	0	0	0
Android MMS	356 392	711	0	0	0
Google Gmail	489 403	0	0	0	0
Contacts Provider	0	0	57 344	0	0
Average	401 273	204	5 213	36 147	249
					Unit: Byte

Table 2. Application states and related files

The size of a program package varies from 92 KB to 951 KB, which would take a few seconds to transfer via a mobile network. The communication cost for synchronizing files in these two categories should not be an issue for typical applications, as the configuration files are quite small, and the contact database should not be updated frequently by the user. Since the contents in such cache buffers are constantly changing and can be re-fetched after being thrown away, our framework labels these files as *no-need-for-sychronization* to reduce the communication costs. As shown in Table 2, the cost for transferring state files are minimal. Web-based applications such as *Google Translate, Android MMS, Gmail* are *state-less* without any state files at all, while the other applications have small state files which are less than 1.5 KB.

Finally, our framework can be used to migrate applications from one physical phone to another physical phone or a computer for a user who would like to continue his/her work after switching to a new phone/computer. The user could use a virtual phone to bridge the migration. In our experiment, we were able to migrate Android applications from a physical phone to a virtual phone, and from the virtual phone to an Android-based notebook computer. For applications which are written entirely in Dalvik, we executed the application natively without translating the binary code and resulted in significantly better performance on the computer. Figure 3 shows the benefit of migrating computer-intensive applications to a faster device. Even with an entry-level Intel Atom processor, the Acer D255 netbook still outperformed the Google G1 phone by 4.9 to 6.4 times.

4 SECURITY ENHANCEMENT FOR VIRTUAL PHONES

While a variety of secure threats are raised by virtual computing environments [14], we believe the operations on a virtual phone hosted by an IaaS provider should still

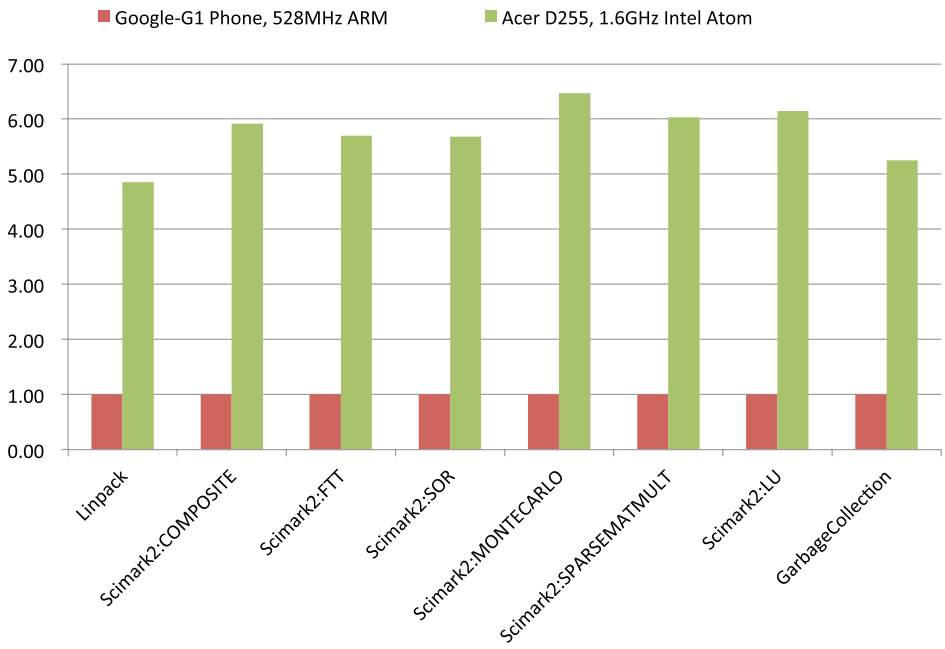

Fig. 3. Normalized benchmark performance

be more secure than having personal data processed and stored on a *software as a service* (SaaS) provider. As the two layers of virtual machines help protect the virtual phone, it is far more difficult for an employee working for the service provider to peek into the virtual phone. To further prevent the intervention from the service provider or attackers, sensitive data in the memory and the storage should better be encrypted.

We envision a new cloud service *Virtual Phone as a Service* (VPaaS) based on the proposed framework. For the user, the framework should take care of the burden of securing the data stored in the virtual phone and protecting the communication traffics between the virtual phone and the physical phone. For the service provider, the framework should work with hardware-based security mechanisms such as the *Trusted Platform Module (TPM)* [24] to enhance the security of the server system. Having a hardware mechanism to store the encryption keys and to perform cryptographic operations is key to avoid data theft from the personnel working for the service provider. For the virtual phone, the hardware mechanism has to be virtualized. For example, IBM's vTPM [9] supports higher-level services for establishing trust in virtualized environments and for supporting live migration. It is possible to integrate such a virtualization scheme into our framework in the future.

Even with a hardware mechanism, the initial set of encryption keys have to be distributed to the user securely. If the provider of VPaaS happens to be the provider of the mobile network, it would be convenient for the provider to store in the SIM card the unique initial keys corresponding to a set of virtualized TPM's. With the keys in the SIM card, the agent on the physical phone may establish a connection using the *virtual private network* (VPN) [22] with the agent on the server to protect the communications between the two agents.

Regarding to the storage on the virtual phone, our prototype includes an encrypted virtual storage to both the physical phone and the virtual phone over a FUSE-based [3] user-space encrypted filesystem, i.e. encfs – encrypted filesystem for FUSE. Since the files are encrypted and hashed, attackers from another virtual machine on the same host or in the middle of the network will be unable to retrieve and manipulate the contents in the files. As mentioned above, it is critical to employ a hardware mechanism such as vTPM to store the master encryption keys and perform the encryption procedure.

The fact that a virtual phone is always connected to the Internet could make the virtual phone a target for a hacker. It is important for the VPaaS provider to hide the virtual phones behind the firewall and not to allow a third party to figure out the association between a virtual phone and its physical phone counterpart. Intrusion detection mechanisms should be adopted by both the service provider and the virtual phone to protect against hackers outside and inside the firewall.

On the other hand, the virtual phone can be used to augment the security of the physical phone. Anti-Virus scan can be a heavy-weight task that can be performed better by a cloud-based service such as *CloudAV* [16] or the virtual phone in our case. The virtualization layer which hosts the virtual phones may be integrated with various security measures such as the detection of vulnerabilities and network attacks [18, 25, 10].

As a case study, we implemented a detection mechanism for the *buffer overflow attacks* (BoA) in our virtual phone framework.

5 RELATED WORKS

It was shown that remote execution saved a large amount of power for mobile computers [20, 11], but partitioning of applications for remote execution has been a challenge for researchers. Spectra [13, 7] proposed to monitor the current resource availability and dynamically determine the best remote execution plan for an application. Cyber foraging [7] used surrogates to improve the performance of interactive applications and distributed file systems on mobile clients. MAUI [12] proposed to reduce the programming efforts by automating program partitioning with the combination of code portability, serialization, reflection, and type safety.

Without application partitioning, methods have been developed to migrate the entire application execution to a remote server. Process migration and virtual machine migration have been two common approaches to migrate execution across

the network. The ISR system [21] emulated the capabilities of suspend/resume functions in a computer system and migrated the system by storing the snapshot image of a virtual machine in a distributed storage system. Zap [17] introduced a POD (PrOcess Domain) abstraction, which provided a collection of processes with a host-independent virtualized view of the operating system, so as to support a general-purpose process migration functionality, but it did not allow live migration. Live migration [15] achieved rapid movement of workloads within clusters and data centers with minimal service downtimes by continuously transmitting the changes in a virtual machine to another system, but at the cost of communication overhead.

6 CONCLUSIONS

In this paper, we presented a framework to automate the creation of a virtual phone and migrates live Android application faster than traditional methods. Compared to a conventional client-server model, our approach does not require the developers to redesign their applications and offers an effective method for the user to control remote execution.

Our preliminary experimental results showed that our virtual phones were capable of intensive workloads and our efficient application-level migration method was suitable for mobile network. The strategies that we developed to reduce the network traffics for application migration and data synchronization are important to the user experience. We also discussed how to enhance the security of a virtual phone service and showed that a virtual phone can be equipped with security measures to protect the system against network attacks. More functionalities, such as remote backup, file sharing, virus scan, and malware detection, could be conveniently integrated into our framework in the future to make a virtual phone more capable and secure. We are designing the API for Android application developers to take advantage of our framework.

Acknowledgement

This work was supported in part by a grant from the National Science Council (98-2220-E-002-020) and a grant from Ministry of Economic Affairs (98-EC-17-A-01-S1-034).

REFERENCES

[1] `http://developer.android.com/guide/basics/what-is-android.html`.

[2] Android Developers web site. Availaible on: `http://developer.android.com/`.

[3] Filesystem in Userspace web site. Availaible on: `http://fuse.sourceforge.net/`.

[4] Google Code Project web site. Availaible on: `http://code.google.com/`.

[5] Acquisti, R.—Gross, R.: Imagined Communities: Awareness, Information Sharing, and Privacy on the Facebook. In: Proceedings of the 6th Workshop on Privacy Enhancing Technologies 2006, pp. 36–58.

[6] Armbrust, M.—Fox, A.—Griffith, R.–Joseph, A. D.—Katz, R. H.—Konwinski, A.—Lee, G.—Patterson, D. A.–Rabkin, A.–Stoica, I.—Zaharia, M.: Above the Clouds: A Berkeley View of Cloud Computing. Tech. Rep. UCB/EECS-2009-28, EECS Department, University of California, Berkeley 2009.

[7] Balan, R.—Flinn, J.—Satyanarayanan, M.—Sinnamohideen, S.—Yang, H. I.: The Case for Cyber Foraging. In: Proceedings of the 10th ACM SIGOPS European Workshop 2002, pp. 87–92.

[8] Bellard, F.: QEMU, a Fast and Portable Dynamic Translator. In: Proceedings of the annual conference on USENIX Annual Technical Conference 2005, pp. 41–46.

[9] Berger, S.—Cáceres, R.—Goldman, K. A.—Perez, R.—Sailer, R.—van Doorn, L.: VTPM: Virtualizing the Trusted Platform Module. In: Proceedings of the 15th Conference on USENIX Security Symposium 2006, Volume 15, USENIX Association, Berkeley, CA, USA 2006, `http://dl.acm.org/citation.cfm?id=1267336.1267357`.

[10] Chen, C. C.—Hung, S. H.—Lee, C. P.: Protection of Buffer Overow Attacks via Dynamic Binary Translation. In: International Conference on Reliable and Autonomous Computational Science 2010.

[11] Chun, B. G.—Maniatis, P.: Augmented Smartphone Applications Through Clone Cloud Execution. In: Proceedings of the 12th Workshop on Hot Topics in Operating Systems 2009, p. 8.

[12] Cuervo, E.—Balasubramanian, A.—Ki Cho, D.—Wolman, A.—Saroiu, S.—Chandra, R.—Bahl, P.: MAUI: Making Smartphones Last Longer with Code Offoad. In: Proceedings of ACM MobiSys 2010, pp. 49–62.

[13] Flinn, J.—Narayanan, D.—Satyanarayanan, M.: Self-Tuned Remote Execution for Pervasive Computing. In: Proceedings of Hot Topics in Operating Systems 2001, pp. 61–66.

[14] Garfinkel, T.—Rosenblum, M.: When Virtual Is Harder Than Real: Security Challenges in Virtual Machine Based Computing Environments. In: Proceedings of the 10th Conference on Hot Topics in Operating Systems 2005, Volume 10, p. 20.

[15] Keir, C. C.—Clark, C.—Fraser, K. H. S.—Hansen, J. G.—Jul, E.—Limpach, C.—Pratt, I.—Warfield, A.: Live Migration of Virtual Machines. In: Proceedings of the 2nd ACM/USENIX Symposium on Networked Systems Design and Implementation 2005, pp. 273–286.

[16] Oberheide, J.—Veeraraghavan, K.—Cooke, E.—Flinn, J.—Jahanian, F.: Virtualized In-Cloud Security Services for Mobile Devices. In: Proceedings of the First Workshop on Virtualization in Mobile Computing 2008, pp. 31–35.

[17] Osman, S.—Subhraveti, D.—Su, G.—Nieh, J.: The Design and Implementation of Zap: A System for Migrating Computing Environments. In: Proceedings of the Fifth Symposium on Operating Systems Design and Implementation 2002, pp. 361–376.

[18] Qin, F.—Wang, C.—Li, Z.—Kim, H. S.–Zhou, Y.—Wu, Y.: Lift: A Low-Overhead Practical Information Ow Tracking System for Detecting Security Attacks. In: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture 2006, pp. 135–148.

[19] Riggs, R.—Waldo, J.—Wollrath, A.: Pickling State in the JAVA System. In: Proceedings of the 2nd USENIX Conference on Object-Oriented Technologies 1996, pp. 241–250.

[20] Rudenko, A.—Reiher, P.—Popek, G. J.—Kuenning, G. H.: Saving Portable Computer Battery Power Through Remote Process Execution. SIGMOBILE Mob. Comput. Commun. Rev. 2, pp. 19–26 (1998).

[21] Satyanarayanan, M.—Gilbert, B.—Toups, M.—Tolia, N.—Surie, A.—O'Hallaron, D. R.—Wolbach, A.—Harkes, J.—Perrig, A.—Farber, D. J.—Kozuch, M. A.—Helfrich, C. J.—Nath, P.—Lagar-Cavilla, H. A.: Pervasive Personal Computing in an Internet Suspend/Resume System. IEEE Internet Computing, Vol. 11, 2007, No. 2, pp. 16–25.

[22] Schmidt, A.—Kuntze, N.—Kasper, M.: On the Deployment of Mobile Trusted Modules. In: Proceedings of the Wireless Communications and Networking Conference 2008, IEEE, pp. 3169–3174.

[23] Schollmeier, R.: A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications. In: Proceedings of the First International Conference on Peer-to-Peer Computing 2001, pp. 101–102.

[24] Sevinç, P. E.—Strasser, M.—Basin, D.: Securing the Distribution and Storage of Secrets With Trusted Platform Modules. In: Proceedings of the First IFIP TC6/WG8.8/WG11.2 International Conference on Information Security Theory and Practices: Smart Cards, Mobile and Ubiquitous Computing Systems 2007, pp. 53–66.

[25] Suh, G. E.—Lee, J. W.—Zhang, D.—Devadas, S.: Secure Program Execution via Dynamic Information on Tracking. In: Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems 2004, pp. 85–96.

Shih-Hao Hung joined the Department of Computer Science and Information Engineering at National Taiwan University as an Assistant Professor in 2005. His research interests include cloud computing, parallel processing, embedded systems, and pervasive applications. He worked for the Performance and Availability Engineering group at Sun Microsystem Inc. in Menlo Park, California (2000–2005) after he completed PostDoc (1998–2000), Ph. D. (1994–1998) and M. Sc. (1992–1994) training in University of Michigan, Ann Arbor. He graduated from National Taiwan University with a BS degree in Electrical Engineering in 1989.

Jeng Peng Shieh received his B. Sc. E. E. and M. EMBA degrees from the National Cheng Kung University, Taiwan, in 1986 and 2007 respectively. He is a Ph. D. candidate in computer science and information engineering of National Taiwan University from 2008. He was a system engineer at BDC Co., Hsinchu, Taiwan (1988–1991), a senior and chief system engineer of RPTI, Taipei (1991–1996), the founder of a software house for designing Internet e-commerce system (1996–2002), the head of R & D department of PROTON Inc., Taiwan (2003–2009). His research interests include Linux-based mobile system design, power and performance optimization, system virtualization, and migration scheme leveraging clustering/cloud computing.

Chen-Pang Lee received the B. Sc. Degree in Electrical Engineering from Fu Jen Catholic University in 1996, and the M. Sc. degree in Control Engineering from National Taiwan University of Science and Technology in 1998. He is currently a Ph. D. candidate in information engineering in National Taiwan University. His research interests include wireless sensor networks and storage systems, especially for data synchronization and security.