用於自攜裝置(BYOD)安全之智慧手機虛擬化技術
Smartphone Virtualization for Bring Your Own Device (BYOD) Security

卓傳育　　林浩澄　　洪茂榮
Chuan-Yu Cho, Hou-Cheng Lin, Anthony

中文摘要

智慧手機及行動上網裝置過去幾年來全球普及率已超過30%，不僅為人們帶來大量的便利智慧生活服務，其不斷提升的運算效能，更逐漸取代傳統筆記型電腦，成為工作上最重要的智慧終端設備。然而員工自攜智慧手機至工作環境使用時，各式各樣自行下載的APP及不同手機中運行的系統服務，亦同時接續入到企業網路內部網路，而造成嚴峻的自攜裝置(BYOD)安全管理挑戰。現行以MDM(Mobile Device Management)嚴格控管智慧終端的方式，仍面臨取捨智慧終端安全管理與使用便利之兩難。智慧手機虛擬化(Smartphone Virtualization)技術以運行彼此相互隔離的安全工作手機環境之能力，達到安全威脅隔離同時可兼顧提供個人自由應用之便利，為BYOD安全的管理帶來新的技術突破契機。本文提出虛擬化智慧手機(Virtualized Smartphone)及虛擬行動基礎架構(Virtual Mobile Infrastructure)兩種方式，可有效滿足在高度安全控管的條件下，仍能兼俱提供自主便利應用服務功能。藉由已可在商用智慧手機中展示的雛型成果，我們已證明智慧手機虛擬化技術可作為BYOD安全的有效解決方案，並且已具備導入到商業環境應用之成熟度。

Abstract

With a more than 30% rapid growth in global smartphone dispersion in last few years, not only are people experiencing exponential increase in convenience from intelligent application services, the continuous breakthroughs on end-device computing have also caused more and more smartphones entering enterprise network environment. Massive and various user-installed apps and services are prone to bring numerously potential security threats directly into enterprise network without any firewall protection. Using MDM (Mobile Device Management) to add strict security policies and monitoring may improve the BYOD security management, but it still encounters a great challenge to strike a balance between security and convenience. In this paper, two smartphone virtualization technologies - Virtualized Smartphone and Virtual Mobile Infrastructure - are proposed to provide a best balance between security management and user convenience. The prototype solutions have been successfully implemented and demoed using generally available commercial smartphones, and the performance evaluation results also support the proposed solutions that are able to provide a close-to-native smartphone user experience.

Key Words

Smartphone Virtualization (智慧手機虛擬化技術)
Bring Your Own Device；BYOD (自攜裝置)
Virtualized Smartphone (虛擬化智慧手機)
Virtual Mobile Infrastructure (虛擬行動基礎架構)

1 · Introduction

With the rapid growth of smartphone devices and mobile applications and services, Bring Your Own Device (BYOD) security not only has become one of the top risks, it has also turned into general data breaches and malware, insider and outsider threat, and advanced persistent threats, according to a latest IT strategy research report by Wisegate [1]. Moreover, BYOD and cloud security have been identified as the most impactful trends to IT security programs. The general BYOD security's challenge could be simply said that many mobile apps are relatively prone to malware compromising the smartphone and yet employees, unaware of that, like to use smartphones to access enterprise network. The various user-installed apps and potential smartphone security breaches are soon becoming potential gateways for malwares to permeate into the company networks, databases, and other systems.

MDM (Mobile Device Management) has been widely deployed as an effective tool to manage BYOD security by installing security monitoring agents onto the smartphones [2]. However, adding strict security-control agent to smartphones incurs strong trade-off between security and convenience. Generally, employee prefers to use their own smartphones freely and without any interventions or privacy offended by the company's agent, whereas the company is obliged to secure every device where the company's data resides. As the result, a secured company smartphone nowadays tends to work only for simple and trivial tasks, and the employee will still continue to bring their own unsecured devices to the working place.

Therefore, the fundamental challenge of modern BYOD security is more than just security manageability, but also finding a way to let employee continue to work effectively and efficiently using a BYOD smartphones without any hassle. Virtualization technology turns out to be a smart solution to this challenge due to its fully isolated nature of virtual machines (VMs). As such, security threats are fully quarantined inside a VM even when the operation system is compromised by kernel rootkits. As a result, running multiple independent and virtual mobile OSes within a single smartphone could provide the best balance between enterprise-level security and personal-device convenience because users could freely operate his/her own smartphone while the potential security damages are well segregated from the other VM.

2 · Way to Smartphone Virtualization

Virtualization technology which began in the 1960s refers to the act of creating virtual version of something, such as computer hardware, operating system, storage devices, and computer network. In the last ten years, virtualization technology has been the key success to the advancement of cloud computing or any other services leveraging cloud computing such as Amazon and Facebook. It is due to the nature of virtualization to be better in resource provisioning, consolidation, convenience, economic, and etc. With the advancement of smartphone's hardware capabilities as well as rapid increase in smartphone adoptions, researchers and companies have been trying to bring virtualization to realm of smartphone to solve various problems [3] [4]. The leading virtualization solution providers such as VMware and Citrix have been working on migrating advanced VDI (Virtual Desktop Infrastructure) onto smart handheld devices and enforcing comprehensive MDM solutions for enterprise BYOD security. On the other hand, many other emerging start-ups such as Sierra, Hypori, Romotium and Cellrox are particularly focusing on incubating smartphone virtualization technologies.

In this paper, two types of smartphone virtualization, Virtualized Smartphone [6] and Virtual Mobile Infrastructure (VMI) [7], are proposed to architect a novel BYOD security solution individually or jointly.

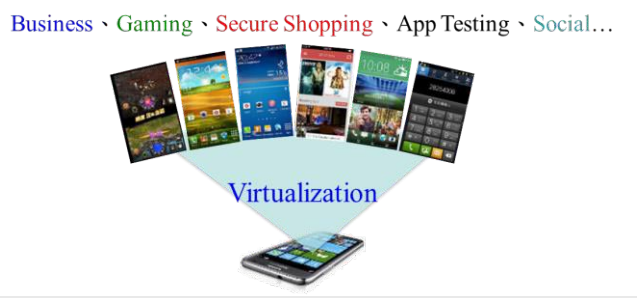


Figure 1 Multiple virtual smartphones running on a single virtualized smartphone solving BYOD security management

Both types of smartphone virtualization are generally depending on whether the requirement to its client device is a thin-client or a smart-rich function smartphone. Virtualized Smartphone requires a powerful handheld device, which allows another Android OS being run on top of the same smartphone in the form of virtual machine(Figure 1). On the other hand, VMI allows us to run the Android OS VM on the remote servers and stream back the VM screen to the smartphone over the internet.

2.1 Virtualized Smartphone

As virtualized smartphone is equipped with a hypervisor, a technique allowing multiple mobile OSes to be run concurrently on the same smartphone, security policies could be applied onto each of guest mobile OSes through hypervisor to address BYOD concerns for each virtual smartphone. For instance, one smartphone now can have four virtual smartphones running i.e a Work, Secure, Personal, Disposable virtual smartphones. As illustrated in Figure 2, a Work virtual smartphone should be strictly locked down using attestation process such as verification of binary of BIOS, OS, applications, and their configurations. In addition, some corporate security policies may restrict some documents to be viewed-only and cannot be sent out directly, and a military grade certification such as the DISA's security technical implementation guide (STIG) may also need to be applied in highly security-sensitive companies.

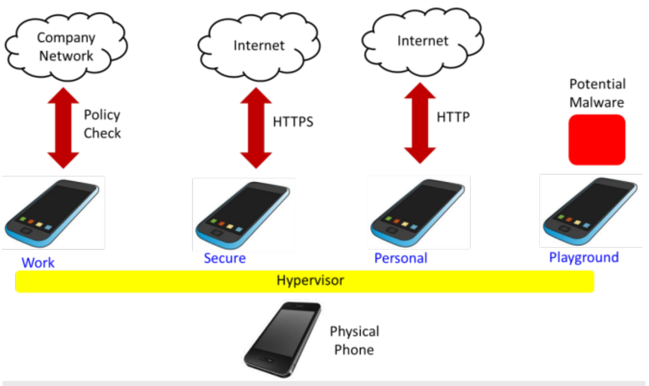


Figure 2 Usage Scenarios for Virtualized Smartphone

When a clean and safe environment for sensitive banking of e-trading applications is required, a Secure virtual smartphone could be used. Hence, whitelisting-based security protection may be applied to make sure no unknown binary is allowed to run and no key loggers exist. For Personal virtual smartphone, it could only be protected using black-listing antivirus solutions, and the users could install and play any kind of apps they wish. Finally, it is possible to let users create a Disposable virtual smartphone to serve as a "Do whatever you want" playground or a sand box for app testing in general.

The major technology challenges in delivering a successful virtualized smartphone include: 1.) Low-overhead hypervisor on smartphones – although today's smartphones are being equipped with more and more CPU power as well as larger memory (up to 4G), it is still a big challenge to efficiently run multiple virtual smartphones at the same time. It turns out that a low-overhead hypervisor on smartphones becomes a major competitive advantage.

With this low-overhead hypervisor, it is then possible to deliver 2.) Same UI fluency inside virtual smartphones and 3.) Seamless context-dependent switching among virtual smartphones. After getting smartphone virtualized, how to enforce the security policies onto each virtual smartphones emerges to be the next major task. Technologies to provide 4.)Virtual machine introspection-based whitelisting, 5.)Display-only file system: having files never leave a server and 6.) Malware detection are among the essential requirements for secure, personal virtual smartphones. Some general ideas for malware detection may include:

(1) Provenance tracking: Is a program properly installed?
(2) Installation checkup: Applications that are NOT properly installed tend to start up in a different way (e.g. in a system start-up script).
(3) Bait-based detection: Files with enticing names are automatically created and placed in random places and are not supposed to be touched by legitimate applications.

To provide display-only file system service and make sure files never leave a server, APP streaming technologies such as the solutions provided by Agawi (acquired by Google), VMFive, mNectar, App.io, AppSurfer and VOXEL are the right answer to this quest by running a mobile App without leaving the enterprise data center so that the data could be securely kept inside enterprise IT system.

2.2 Virtual Mobile Infrastructure (VMI)

APP streaming could be viewed as a simplified version of Virtual Mobile Infrastructure (VMI) which streams only a single App instead of the entire virtual smartphone VM activities.[8][9] As for BYOD security, VMI might be the best, cross-platform and thin-client solution that is analogous to the VDI for Desktop PCs.

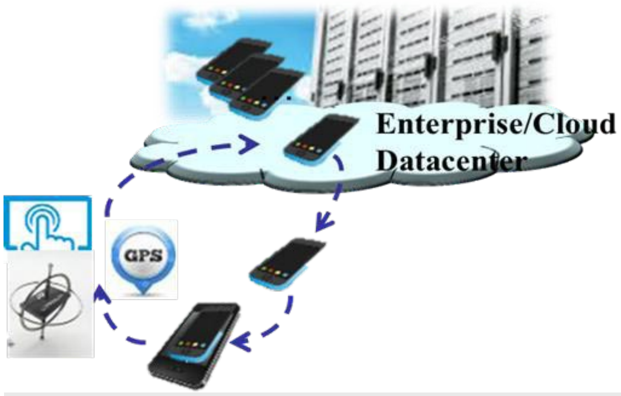


Figure 3 Virtual Mobile Infrastructure keeps apps and data in enterprise's datacenter and thus no data leakage risk

Figure 3 presents a basic concept of VMI which includes both mobile desktop streaming and local sensors redirection such as GPS, gyro and multi-touch events. The VMI is particularly useful in dealing with applications that have exhaustive computing power consumption e.g. CAD/CAM viewers and editors; network usage for large file, video and rich web content browsing; and high data-security needs such as centralize data management. Most importantly, the best part of VMI is really the "no risk at all of device stolen or lost". However, the access bandwidth's variance and limitation of enterprise network are still the major road block for better VMI user experience. In addition, rich-sensor devices and the need for instant device interaction also bring significant technology challenge when developing a commercial-grade VMI solution.

Furthermore, in the VMI server side, efficiently running Android for x86 servers plays another key role and how to ensure various apps could be running properly without any modification could be a potential challenge as well. Moreover, the client sensing devices such as touch sensor, gyro sensor, GPS, camera, and others may also need to be equipped onto the Android VM using respective virtual device interfaces, which makes further difficulties on both device emulation and instant result streaming.

3 · Virtualized Smartphone

As the Android kernel is a modified Linux kernel, existing virtualization solutions in Linux can be easily ported to Android environment. The difference of an Android kernel is mainly at the additional fast IPC mechanism for reducing the effort of application / framework service communications, and wake-lock mechanism for fine-grain control of power saving by applications.

The proposed virtualization solution used in this paper is KVM (Kernel-based Virtual Machine) kernel module and QEMU (Quick Emulator) software. The KVM relies on Intel VMX or AMD SVM hardware to configure a virtualized CPU, virtualized memory and to control the entry/ exit to/from VM mode. The KVM provides a KVM API for user level process to create a VM and to assign its associated VCPU, memory and the run in VM mode. The QEMU process calls the respective KVM APIs to initial a virtual machine and to emulate its I/O devices activities when virtual machine access device through IO port. In addition, QEMU is also in charge of deliver infrastructure features such as network address translation (NAT) for providing network access to VM, network port forwarding to export the service access point of VM to public network and VNC (Virtual Network Computing) server that provides remote console access to the VM.

The software architecture of proposed Android virtualization application is depicted in Figure 4, and it is similar to Linux hypervisor software: KVM module runs in kernel mode, QEMU component runs in user mode and one extra virtualization app component runs on top of Android Davlik JVM.

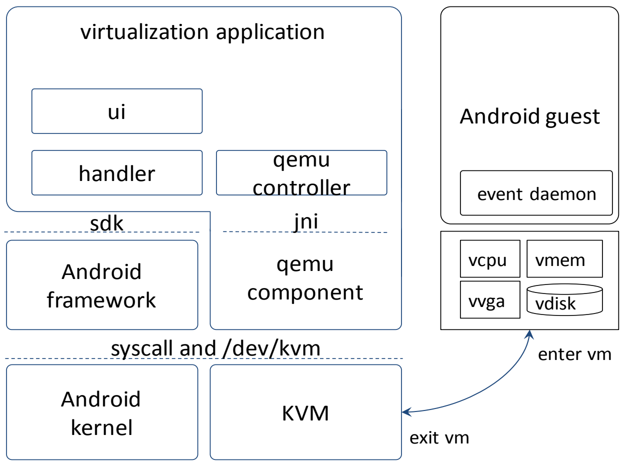


Figure 4 Software Architecture of Android Virtualization

As the Android framework is a Java based execution environment whereas QEMU depends on C libraries, it requires JNI (Java Native Interface) to communicate between QEMU and the Android app. In Figure 4, the Android framework interacts with app component through Android SDK interface, and the app component contains one QEMU controller that calls QEMU component's main function through the JNI interface. The VM configurations are in forms of function parameters, passed to start VM function, including number of vCPUs, size of memory, disk image file name and the network settings. The QEMU then calls KVM kernel module through KVM ioctl command to "/dev/kvm" device for creating the virtual machine and then call kvm_run ioctl command to enter the VM execution mode.

3.1 Virtual Smartphone VM APP

Since the Android framework is written in Java and executed in a JVM(Java Virtual

Machine), the Android application are also in Java byte code to access the API and service of framework. The top-half of virtualization application is also written in Java and includes the following three main packages: UI, handler and QEMU controller. The UI package's main purpose is to handle UI event, refresh display of virtual machine and to call other packages while getting click events. The QEMU controller is invoked by UI package and is responsible to provide APIs to control or to query QEMU states as well as to get pointer of vga frame buffer and its dirty status. The QEMU module is finally executed by QEMU controller via JNI interface in order to access the memory block or to call functions in C code level.

The handler is responsible to forward events in host into VM's event daemon via SVMP (Secure Virtual Mobile Platform) protocol.[10][11][12] The SVMP protocol is borrowed from SVMP project [4], an open source VMI project, which contains definition of various phone events. For example: the multi-touch event, GPS location event and phone rotation event. These events will be received by handler package and be forwarded to Android VMs accordingly. Reversely, for events generated inside Android VMs are also delivered by event daemon to handler package. For example, the notification messages in Android VMs have to be encapsulated into SVMP messages and deliver to handler package to display it on the host's notification board.

3.2 Display Optimization

A VM run inside the QEMU module has output its display on to an emulated VGA (Video Graphic Array) card on host environment. User may either use remote display protocol or SDL library to get the VGA frame buffer and display it on to the physical display device. However, these two approaches are both not feasible on an Android phone because Android environment does not support SDL library and the remote display protocol's performance is too slow due to the unnecessary compress and frame buffer memory copy operations. As a result, a better approach for virtual machine display is proposed with the following optimizations:

(1) directly access the frame buffer memory that resident in virtual machine.
(2) using host phone's 3D chip that accelerate display speed.

In order to directly access the frame buffer, it is necessary to look into the QEMU's vga card emulation code and add a JNI function get_framebuffer(), to return the pointer of vga's vram frame buffer. The UI package's display refresh code, runs in Java byte-code level, calls this JNI function to get the frame buffer in C level, and then update it to phone's screen accordingly.

The UI package relies on the OpenGL 3D library to create a simple 3D world with the following three objects: uniform light source, wall for painting VGA frame buffer on top of it and a viewpoint in front of the wall. The UI package first periodically paint the content of VGA frame buffer onto the wall, and then a user on the viewpoint, sitting in front of the wall, will see the VM's VGA output from the wall just similar to the audience watch movies in the theater. The "texture" properties of a 3D object are then periodically refreshed through binding pointer with VM's VGA frame buffer.

For better performance, one dedicated thread is create to constantly update VGA frame buffer onto wall and to run OpenGL render scripts to generate the viewable scenes. The steps of frame buffer fetch, update and render are depicted in Figure 5.

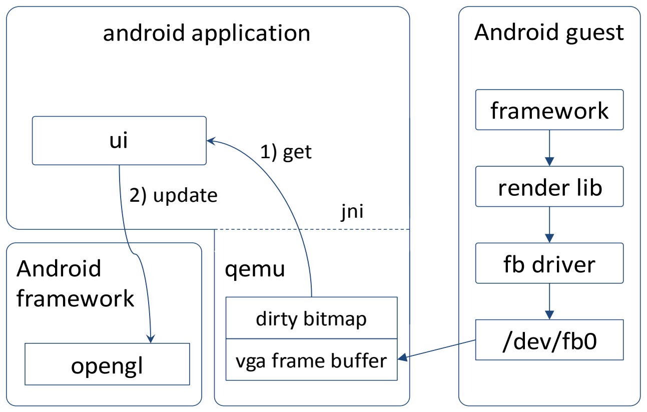


Figure 5 Design of Display Acceleration

As most of the time the VM's display is unchanged, the update and render activity of the unchanged frame should be skipped for better performance and efficiency. As QEMU support a dirty memory tracking feature, we further leverage this feature to enhance display experience. If the VGA frame buffer is not marked as dirty, its update is skipped. Otherwise, current frame is updated into OpenGL, and then is rendered follow by a reset memory dirty flag reset operation.

Since the implementation of dirty memory tracking APIs is based on hardware MMU's writing protection feature, the cost to trace memory dirty is relative cheap whereas the results of skipping frame update could significantly reduce the CPU and GPU loads.

3.3 Experimental Results

In this paper, a working prototype of a virtualized smartphone has been successfully built. Leveraging Android-x86 project[13], an open source project that port Android AOSP image to x86-based architecture machine and maintained by Taiwanese software engineer, as the guest OS of the virtual machine running on a commercial phone ASUS Zenfone2 which has been rooted and bootloader unlocked, our project has a close collaboration and active support from the related parties. In summary, our project leverages QEMU, an open source hypervisor, and SVMP for the sensor and I/O redirection mechanism. Lastly, our display acceleration mechanism significantly improved the time-to-display overhead.

The features supported in current prototype includes:

(1) OpenGL direct display. Fast display by directly access and display the frame buffer of the guest VM.

(2) Single & multi-touch. Multi-touch events are able to be forwarded to the VM.

(3) Rotation event. When the host-screen is rotated, the VM will also rotate its screen.

(4) Dial-out Call. Upon attempt to dial a number from inside the VM, user will be redirected to dial-out app on host phone.

(5) GPS Location. Every change in the host GPS data will be sent to the VM, allowing the VM to have similar behavior as the host.

(6) Audio Redirection. The VM is able to directly use the host audio HAL (Hardware Abstraction Layer) to playback the sound.

Our display acceleration is able to reach average of 27~28FPS (Frame per Second). Together with the six features mentioned above, these are primarily factors that set us apart and bring us on the same position as existing indirect competitors (e.g. Samsung, VMware MVP). Furthermore, ongoing works are being done to improve the 3D graphic support and acceleration on the VM

Table 1 Host Overhead Measurement

Overhead of the host when	Memory (MB)	Energy (mAh) per 10 min	Current (mA)
(a) KVM turned off	854.81	3.046	18.28
(b) KVM turned on with no VM is running	858.37	3.146	18.88
(c) KVM turned on with one VM is running	866.12	17.06	102.36

Table 1 presents the host memory and power consumption overhead of current prototype for different cases before and after KVM (hypervisor) is turned on as well as before and after a VM is being run. The memory usage's range is around 1181 to 1308 MB while the electric current indicates an increment from case (a) to (c). The case (c) which is the case when one VM is running shows current drawn of 102.36 mA, yet the case (a) which is the baseline has already used up 18.28 mA. Therefore, host drawn additional current up to 102.36 - 18.28 = 84.08 mA as an impact of running one VM.

Table 2 Individual Overhead Measurement

Individual Overhead	Memory (MB)	Energy (mAh) per 10 min	Current (mA)
(a) A browser app	270	54.4	326.4
(b) An idle VM	566.5	31.25	187.5
(c) A VM running an active browser app	568.5	72.55	435.3

Comparison of a browser app running on host smartphone, an idle VM, and a VM running active browser app inside it is presented in Table 2 for the reader to have some intuitive overhead estimation of running one VM in our prototype. As in the second column, memory usage of a VM can vary depending on the initial assignment, in this case it is assigned 512MB initially. Interestingly, case (a) and (c) gives us comparison of a browser app and VM running browser app which indicates that running a VM only requires approximately 33% more resource than in running browser app.

Table 3 Virtualization Storage Footprint

	KVM module (kB)	QEMU executable (kB)	Zenfone2 kernel w/o KVM (kB)	Zenfone2 kernel w/ KVM(kB)
Disk Usage	896.1	32,710	12,816	13,047

As shown in Table 3, there are four major storage footprint incurred by our virtualization technology. The first column is the total storage occupied by kvm modules, namely, kvm_intel.ko and kvm.ko. Next column, a modified QEMU executable file is required to run each VM image in Android environment which used up 32.7MB of disk. This size is relative large due to porting effort done in porting QEMU from Linux to Android environment so that some dynamic libraries were made into one static executable file. Importantly, the last two columns show the size of Zenfone2 kernel images before and after KVM was built into the kernel.

4 · Virtual Mobile Infrastructure

4.1 Design Principle and Architecture

The key principle of VMI is that the mobile VMs are residing at the remote server and users connect to the VM and stream back the screen over the network. As such, in BYOD solution, client smartphone doesn't suffer from heavy overhead of running VM or running some device management processes.

Leveraging SVMP (Secure Virtual Mobile Platform), a free and open source project, our

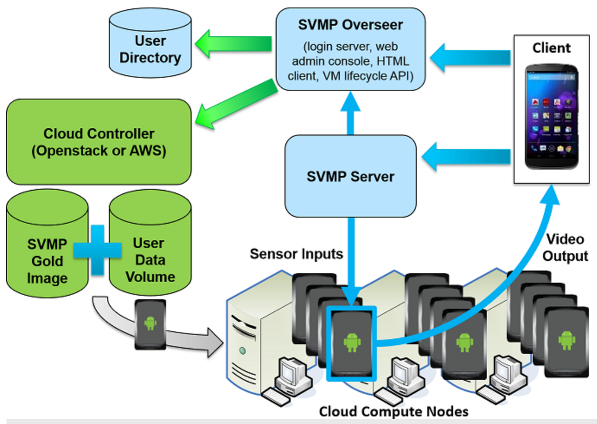


Figure 2 SVMP VMI Architecture

early working prototype for VMI has been achieved. Figure 6 shows the architecture and the workflow. Firstly, the client-end connect to the SVMP Overseer through a native app for authentication and afterward the connection is redirected to SVMP Server which is acting as the proxy and also redirecting the connection to the designated VM. Lastly, the connection is received by the SVMP daemon which is running inside the VM. Once the connection to VM is established, the VM sends the video steam via WebRTC (Web Real-Time Communication) protocol.

Furthermore, SVMP is also designed to be easily deployable as an application on top of existing virtualization systems and public/private clouds like OpenStack.

4.2 Implementation Details

Unlike traditional remote desktop applications designed for keyboard and mouse input, SVMP lets users interact naturally with remote applications using native mobile inputs like multi-touch, location, and sensors for better interaction between client app and remote VM. Thus SVMP has added a set of virtual input devices, video streaming output, and some other customizations to enable a rich remote access experience. More details are shown in figure 7.

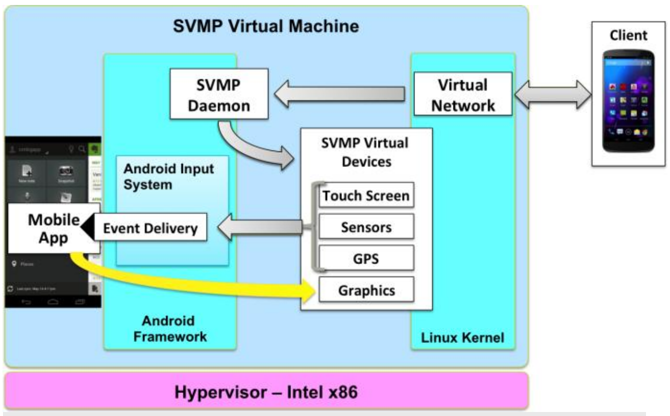


Figure 3 SVMP Virtual Device Structure

As in figure 7, modification of the VM (Androidx86) source code is required in achieving a more natural, rich and user-friendly interaction between the remote client and the VM. The SVMP Daemon, Touch Input, Sensors, Location update, Intent and Notification, Video are the components that require modification.

(1) The SVMP daemon is the background service running in the VM that is the primary entry point of client user input to the VM.

(2) The touch input events which are generated on the client app are forwarded to the VM using protocol buffers and are handled by SVMP daemon by injecting them to the VM as they are received.

(3) Sensor events generated on the client app are forwarded using protocol buffers by SVMP daemon to the local listening socket on the VM. Then, the SVMP HAL module libsensors listens to the socket and processes the actual sensor events.

(4) Location events are communicated both ways between the client app and the VM using protocol buffer messages. Any subscriptions intents requested by apps from the LocationManager on the VM are passed back to the client app. The client then passes the Location information to the VM.

(5) Exchange of intents between the SVMP client app and the VM is supported by SVMP. For example, when users try to call a phone number in the VM, the SVMP client app is able to receive the ACTION_DIAL intent. Furthermore, notifications received from the VM will also be shown from the client app.

(6) Video output of VM, at the lowest level, will be displayed to a Virtual FrameBuffer (VFB) device from Linux kernel instead of to a real video device.

Frames written to VFB are copied after the screen has been fully updated and then fed to the WebRTC subsystem frame by frame for video encoding and streaming. On the receiving end, the source stream from the VM simply appears as a standard video streaming source. Thus, the client app just handle it as a standard WebRTC video stream.

4.3 Experimental Results

Our experiment with SVMP project was done by using Androidx86 VM image that has been integrated with SVMP daemon. The result is fairly good. In terms of the features, we have verified the multi-touch, screen rotation, GPS sensor forwarding and dial-out. In terms of sleekness, it still highly depended on the internet connection speed. Moreover, some modifications such as fake IMEI and WiFi-MAC address number provider have been implemented. However, there is still a long road to go, issues such as 3D virtual GPU support or pass-through still be a challenging issues and also how to make the VM seems to be as similar as physical phone remains to be challenges for us.

5 · Conclusion

In this paper, we have implemented a prototype of Virtualized Smartphone on a commercially available smartphone and measured performance, which interestingly is close to native user experience such that it is considered to be mature enough for delivering a commercial-grade BYOD virtualized smartphone. Virtual Mobile Infrastructure prototype which is derived from SVMP project, have also portrayed an almost ready framework to deliver a thin-client BYOD solution today. With smartphone virtualization technology, we are trying to enable brand mobile device vendors a way to cut-in enterprise BYOD security market, and to create unique smartphone features other than only UI and app customization available today. Our future works will focus on continuing to maximize runnable apps inside an Android VM or a non-virtualized ARM device to make sure users get exactly the same experience as the one from a native smartphone. For instance, GPU virtualization is still not available for Android VM; real-time multimedia streaming such as Camera, Voice sensors or live media playback are also challenging issues in most enterprise network infrastructures, the limited end-device computing and memory resources.

Reference

[1]Elden Nelson,“ BYOD and cloud are top data breaches and malware risks, survey shows,” http://www.csoonline.com/article/2906359/data-breach/byod-and-cloud-are-top-data-breaches-and-malware-risks-survey-shows.html, Apr 6, 2015.

[2]Ji-Eun Lee, Se-Ho Park and Hyoseok Yoon,” Security policy based device management for supporting various mobile OS,” Computing Technology and Information Management (ICCTIM), 2015 Second International Conference on Date of Conference, pp.156-161, 21-23 April 2015.

[3]V. Munshi, Virtualization: Concepts and Applications, The ICFAI University Press, 2006.

[4]“Virtualization” [online]. Available: https://en.wikipedia.org/wiki/Virtualization

[5]Xiaoyi Chen,“ Smartphone virtualization: Status and challenges,” Electronics, International Conference on Communications and Control (ICECC), pp. 2834 – 2839.

[6] Shakuntala P. Kulkarni1, Prof Sachin Bojewar,” Survey on Smartphone Virtualization Techniques,” International Research Journal of Engineering and

Technology (IRJET), vol. 02, Issue: 04, pp.371-376, July-2015.

[7] Justin Marston," Virtual Mobile Infrastructure: Secure the data and apps, in lieu of the device," http://www.networkworld.com/article/2937789/mobile-security/virtual-mobile-infrastructure-secure-the-data-and-apps-in-lieu-of-the-device.html, Jun 18, 2015.

[8]Eric Y. Chen and Mistutaka Itoh," Virtual smartphone over IP," IEEE International Symposium on World of Wireless Mobile and Multimedia Networks (WoWMoM), 2010, pp. 1 – 6.

[9]Masashi Toyama, Shunsuke Kurumatani, Joon Heo, Kenji Terada, and Eric Y.Chen," Android as a server platform," IEEE Consumer Communications and Networking Conference (CCNC), 2011, pp.1181 – 1185.

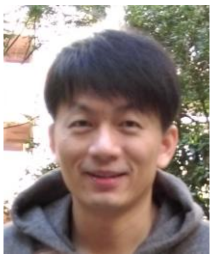
[10] "Virtual Smart Phones in the Cloud" [online]. Available: https://svmp.github.io/index.html

[11] "SVMP System Design and Architecture" [online]. Available: https://svmp.github.io/architecture.html

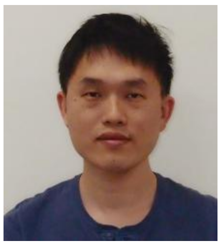
[12] "Open source SVMP project": https://github.com/SVMP

[13] Android-x86.org, http://www.android-x86.org/

Authors



卓傳育 (Chuan-Yu Cho) received his Ph.D. degree in Computer Science from National Tsing Hua University, HsinChu, Taiwan, in 2006. He is now a senior software engineer and manager of Datacenter System Software (Div-F) division hypervisor team, Information Comm. Research Lab at ITRI. His research interests include cloud computing, virtualization technology, cyber security, image processing, video coding and streaming.



林浩澄 (Houcheng Lin) received his Master degree in CSIE from National Chiao Tung University on 1997. He is now Senior software engineer of Datacenter System Software (Div-F) division hypervisor team, Information Comm. Research Lab at ITRI.

洪茂榮(Anthony) received his B.Sc. degree in CS from National Chiao Tung University in 2015. He is now a member of Datacenter System Software (Div-F) division hypervisor team of Information Comm. Research Lab (ICL) at ITRI