ViSP: A Cloud-based Virtual Smartphone Platform

Jiajun Wang[1,*], Jigang Wang[2], Peng Yang[3], Zhicheng Ma[3], Lei Zhang[3] and Gang Wang[3]
[1]800 Dongchuan Road, Shanghai, China, 200240
[2]800 Tian Fu Da Dao Zhong Duan, Gaoxin District, Chengdu City, Sichuan Province, China, 610041
[3]629 East Xijin Road, Qili River, Lanzhou City, Gansu Province, China, 730050
[*]Corresponding author

***Abstract*—Smartphones have become increasingly ubiquitous in our daily life. However, mobile devices are less powerful than traditional devices like desktops and laptops. And the hardware resources of each mobile device also vary widely which leading to different user experiences of smartphone users. This paper proposes ViSP, a cloud-based virtual smartphone platform. By leveraging virtualization technology, we provide virtual smartphones, which are deployed in the cloud. Users can use apps remotely on the virtual smartphones, ignoring the limits of physical devices. Experiments show that the bandwidth cost is about 100kBps in average using zlib encoding, which is suitable and reasonable in 4G network.**

Keywords-virtualization; android; remote display; cloud computing

I. INTRODUCTION

The number of smartphone users is growing rapidly these years. More and more people spend time using smartphone instead of laptop or desktop computer for its portability and connectivity.

However, hardware resources of smartphones are typically very limited compared to those of traditional computers, such as central processing unit (CPU), memory, storage and battery. And mobile app developers are required to take these limitations into consideration. Besides the difference in hardware resources, software stack also varies between devices. There are so many different mobile Operating Systems (OSs) currently such as iOS, Android, Windows, etc. And more and more mobile OSs are appearing like Tizen, Sailfish. The discrepancy between different devices leads to diverse user experiences. Some computation-intensive apps may not run smoothly on low-end mobile phones. And some apps may not come with a specific mobile OS due to lack of mobile developer in the software company.

Meanwhile, virtualization has been thoroughly researched these years and applied to industry to manage resources more effectively. Desktop virtualization [1] has been used in organizations and companies to ease the burden of maintaining computers and to improve the resource utilization. There are typically enormous desktop computers and servers in an IT organization. Managing so many computers and servers could be a hard burden. Using virtualization, all the computers can be managed on the cloud side while creating new instance or upgrading them is more convenient. In the same way, this method could also be applied to smartphones to provide a unified experience for mobile users and easy to maintain.

In this paper, we propose ViSP, a Cloud-based Virtual Smartphone Platform, using thin-client computing with mobile phones to provide virtual mobile OSs to end users. By running OSs and apps remotely, the gap between different devices could be narrowed.

The remainder of this paper is structured as follows. Section II describes the design of our platform. Section III introduces our implementations in details. Section IV reports the evaluation results of the proposed platform. Section 0 presents the related research work about and thin client computing and virtual smartphone. Section VI concludes the paper and discusses the future work.

II. ARCHITECTURE DESIGN

In order to provide good user experience to mobile phone users, we propose ViSP in a cloud environment to leverage virtualization technology.

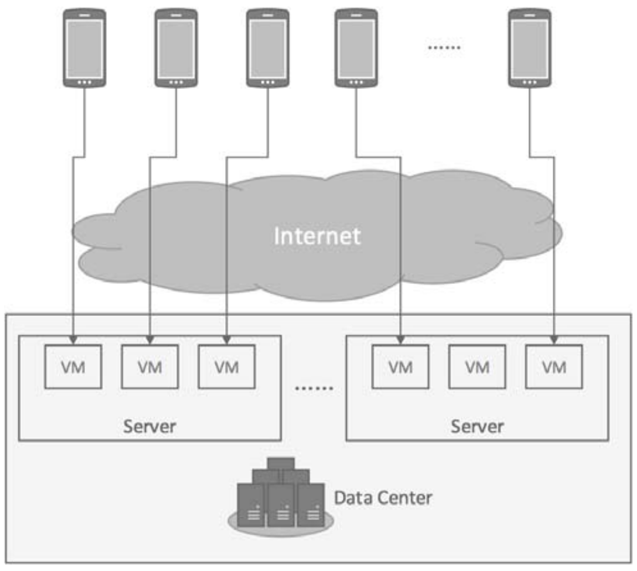

FIGURE I. OVERVIEW OF VISP

Figure I shows the high-level overview of ViSP. Virtual smartphones are running on the servers in the data center. A user can own one or more different virtual smartphones. He can access the virtual smartphone using ViSP client as long as he has internet access.

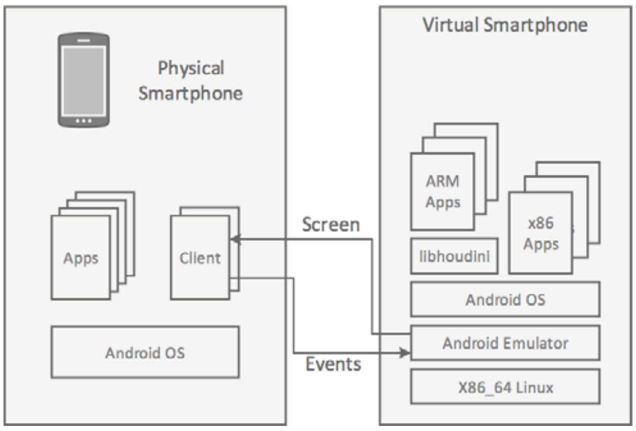

FIGURE II. DETAILED ARCHITECTURE OF VISP

Figure II presents the detailed architecture of our system. The ViSP client is also an app on smartphone, which receives the screen updates from the server and shows it on the physical smartphone. Thus, the user is expected to use the virtual smartphone as a separate mobile OS from the original OS. And the client intercepts all the touch events and then sends them back to the server so that the virtual smartphone can get user input from the physical device.

To make the platform easy to use and maintain, we must ensure the transparency of the architecture. Our platform should not require any modifications to the virtual smartphone OSs. All work should be done at the hypervisor layer so that different mobile OSs can run on our platform as long as they can be virtualized on some hypervisor. This means that we must intercept the display of virtual smartphone and send in the control events at the hypervisor level.

III. IMPLEMENTATION

We have implemented a prototype using Android as the server-side virtual smartphone OS. Android emulator has been provided in Google's Android SDK and both Android system and the emulator are open source in Android Open Source Project.

A. *ARM Native Code Support*

Android apps are typically written in Java and run on Android runtime, Dalvik before Android 4.4 and Android Runtime (ART) since Android 5.0. However, Android also allows developers to implement parts of their apps in native C/C++ code in order to speed up some CPU-intensive programs. For instance, some game engines (e.g., Unity, Cocos2d-x) are written in native code for performance and portability.

In the early versions of Android, ARM is the only platform which is officially supported by Google. Android-x86 is an unofficial initiative to port Android to AMD and Intel x86 chips.

Though x86 and MIPS chips are officially supported by Google nowadays, there are still some apps which only contain ARM native codes. To solve this problem, Intel has created a compatibility layer named libhoudini. This library acts as a binary translator, reading in the ARM instructions and converting them into the corresponding x86 instructions on the fly. So most apps could run on an x86 Android as normal.

B. *Screen Updates*

Some traditional remote display protocols use client-pull mode to update the screen image. This means that the client is responsible for handling the screen updates. Once the client thinks the screen should be updated, for example, some touch events have been fired or a fixed time interval has passed, it will request a new screen image from the server. Hence, the latency is a bit high, which is one Round Trip Time (RTT). While in server-push mode, the server sends the updates once the screen is changed on the server side. And latency is half of RTT, which is better than client-pull mode.

In server-push mode, the server acts as the producer while the client is the consumer. The server appends the screen updates into a buffer queue. The client takes out the updates from the queue. Unfortunately, if the network connection is slow, the producing speed is faster than the consuming speed, user experience will suffer. Once the client could not consume the screen updates in time, the buffer queue will become longer, leading to lag on the client side. Therefore, the server puts a sequence number in each screen update packet, and client sends back an acknowledgement packet to indicate it has received a specific update. If the server produces lots of updates while the client has not responded yet, the update packets will be dropped to avoid the lag.

IV. EVALUATION

The experiments are conducted in a separate network environment. The server has a 3.1GHz Intel Core i5-3450 CPU and 8GB system memory. The client app runs on LG Nexus 5. The virtual smartphone OS is Android 5.1 running on our modified Android emulator. The resolution of virtual Android is 320x480.

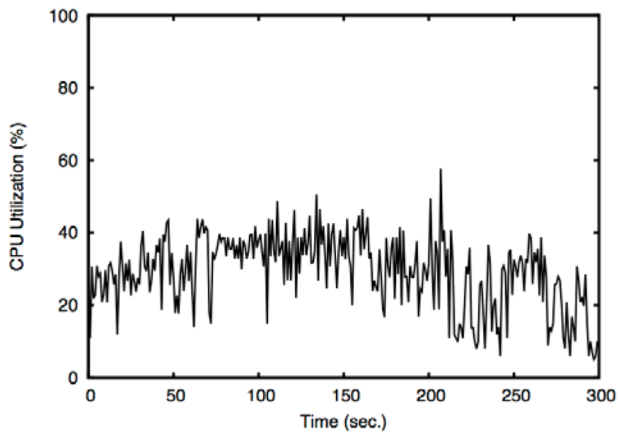

FIGURE III. CPU UTILIZATION ON SERVER

The CPU utilization of the server is presented in Figure III. The average CPU utilization is 29.25% on one core. A quad-core processor like the one we use could contain up to 10 virtual Android on one server.

Running virtual Android onthe serverr can greatly reduce the CPU burden of physical device. All apps could run on a server whose CPU is considered more powerful than the CPU on mobile devices. However, screen updates must be transferred through network. The bandwidth cost must be considered since users may be using 3G/4G. They may be charged according to their data traffic.

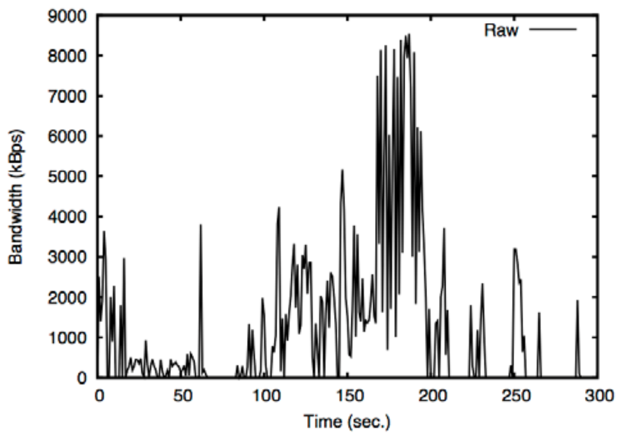

FIGURE IV. BANDWIDTH COST USING RAW ENCODING

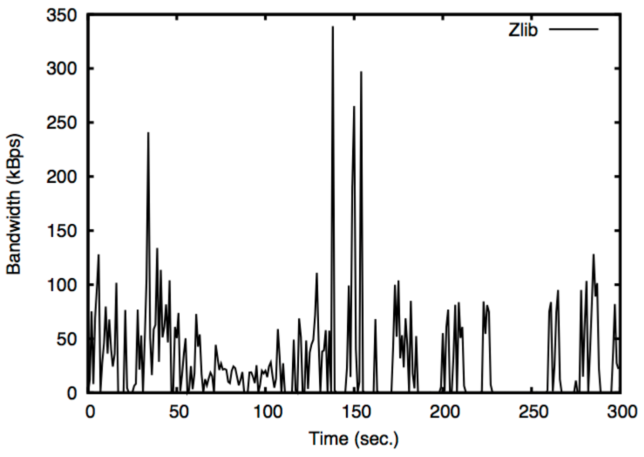

FIGURE V. BANDWIDTH COST USING ZLIB ENCODING

We have recorded the bandwidth cost during a 5minutes period as shown in Figure IV and Figure V. Figure IV shows the bandwidth cost if the screen is directly sent to the client. From the third minute, we start a game on the virtual Android, so the bandwidth has become rather high in the figure. Using zlib encoding, we can greatly compress the screen images. Figure V refers to the bandwidth cost under zlib encoding. Even playing game does not consume too much network bandwidth. The average bandwidth cost of raw and zlib encoding are 1199.2 kBps and 79.1 kBps. The bandwidth could be further reduced if a lossy compression algorithm is used like JPEG.

V. RELATED WORK

Virtualization allows multiple virtual machine instances on one physical machine simultaneously. Xen [2] is proposed to virtualizes x86 through paravirtualization. And KVM [3] is a full virtualization solution on x86 linux hosts.

Thin-client computing has been researched for a long time. The X system [4] uses a C/S architecture to separate the UI and program. Display commands are forwarded from an X client to an X server to be drawn on screen. VNC [5] works at framebuffer level. The VNC client will ask the server to send the screen updates.

THiNC [6] virtualizes the display at device driver. It translates the display commands into some low-level commands to be processed at ThiNC client. A mobile client is presented so that users can remotely control a PC on a smartphone. Lamberti [7] uses MPEG video streaming to transfer the remote screen to the client in order to reduce the bandwidth cost. Virtual Smartphone over IP [8] provides a similar architecture using Android-x86 and VNC, while adding sensors support.

VI. CONCLUSION AND FUTURE WORK

This paper presents ViSP, A Cloud-based Virtual Smartphone Platform. Using virtualization, users can create virtual smartphones in the cloud and connect to the virtual smartphones using a client app on their mobile phones. The experiments show that ViSP demands low bandwidth with zlib compression and can provide good user experience.

We plan to add multitouch and sensor support to better use the capabilities of physical devices. Furthermore, we plan to migrate to KVM or Xen as the server platform since they provide some advanced features like management APIs, live migration and so on.

REFERENCES

[1] Lai, Guangda, Hua Song, and Xiaola Lin. "A service based lightweight desktop virtualization system." Service Sciences (ICSS), 2010 International Conference on. IEEE, 2010.

[2] Barham, Paul, et al. "Xen and the art of virtualization." ACM SIGOPS Operating Systems Review 37.5 (2003): 164-177.

[3] Kivity, Avi, et al. "kvm: the Linux virtual machine monitor." Proceedings of the Linux Symposium. Vol. 1. 2007.

[4] Scheifler, Robert W., and Jim Gettys. "The X window system." ACM Transactions on Graphics (TOG) 5.2 (1986): 79-109.

[5] Richardson, Tristan, et al. "Virtual network computing." Internet Computing, IEEE 2.1 (1998): 33-38.

[6] Baratto, Ricardo A., Leonard N. Kim, and Jason Nieh. "THINC: a virtual display architecture for thin-client computing." ACM SIGOPS Operating Systems Review 39.5 (2005): 277-290.

[7] Lamberti, Fabrizio, and Andrea Sanna. "A streaming-based solution for remote visualization of 3D graphics on mobile devices." Visualization and Computer Graphics, IEEE Transactions on 13.2 (2007): 247-260.

[8] Chen, E. Y., and M. Ito. "Virtual Smartphone over IP. Montreal." QC, Canada: IEEE WOWMOM (2010).