EAASI PRESERVATION OF MOBILE APPLICATIONS

Progress with the long-term preservation of access to mobile applications using the EaaSI platform

Euan Cochrane
Yale University Library
Country
euan.cochrane@yale.edu
0000-0001-9772-9743

Jurek Oberhauser
OpenSLX
Germany
jurek@openslx.com
0000-0003-4542-7959

Rafael Gieschke
University of Freiburg
Germany
rafael.gieschke@rz.uni-freiburg.de
0000-0002-2778-4218

Abstract – Mobile devices have revolutionized computing and democratized access to it. The applications we use on our mobile devices play a critical role in shaping our online experiences, our culture, our politics, and our access to information. Mobile applications are also widely used for data gathering and asset management in many domains from scientific research to infrastructure maintenance. With such a wide-reaching impact it is critical that the preservation community is able to maintain access to mobile applications for future generations. In this short paper we outline progress in using the Emulation as a Service Infrastructure (EaaSI) platform to run obsolete versions of the Android operating system in virtualization and emulation in order to maintain access to mobile applications. We also detail the current limitations of virtualizing and emulating mobile devices and provide a list of future challenges to address as we move forwards with ensuring long-term access to this essential part of our history.

Keywords – emulation, mobile, apps, applications

Conference Topics – From Theory to Practice; Immersive Information

I. INTRODUCTION

Within the Emulation as a Service Infrastructure (EaaSI) platform users can already run some versions of the Android operating system using the existing QEMU emulator. Since Android is a variant of the well-supported Linux-based operating system family, Android versions made for the IBM PC platform using the x86(-64) architecture [1] can generally run on the modern versions of QEMU with no special customizations being required. In addition, Android comes with a driver for optical drives so existing workflows in EaaSI that allow for installing new software via an optical drive work with these versions of Android, also with no customization required. However, the configuration of QEMU to support desktop Linux-based operating systems and the way it is integrated within the EaaSI user interface assumes a limited number of inputs and outputs. Mobile devices generally have quite different input and output methods relative to desktop computers. For example, mobile devices often accept touch input, GPS sensor input, gyroscopic sensor input, and many other mobile-oriented inputs, and will output mobile-oriented outputs like vibrations, device-based-sharing protocols, and other mobile-specific outputs. Within EaaSI there is significant work to do to integrate these mobile-specific inputs and outputs into the EaaSI interface and workflows. In addition, we need to add options for simulating inputs like GPS coordinates and health sensor data.

iPRES 2023: The 19th International Conference on Digital Preservation, Champaign-Urbana, IL, US.

https://hdl.handle.net/2142/121110

1. Technical Challenges

1. *Emulation and Virtualization of Mobile Devices*

The Android OS itself is an open-source project [3], however most Android versions that are used on handheld devices are modified by the device manufacturer to support device-specific operations. These devices overwhelmingly depend on the ARM(64) architecture. While ARM emulators exist, emulating ARM-based Android images becomes a tedious task, as both performance and user experience suffer. Virtualization becomes a necessity if suitable user experience should be provided. As the desktop and server-based hardware on which the emulators run is usually x86(-64)-based, hardware acceleration and virtualization cannot be provided for images with full ARM emulation. Additionally, ARM emulation is more complex to set up as, in contrast to the x86(-64) architecture with the IBM PC platform, there is no universal ARM-based platform yet but many different platforms[1], i.e., combinations of a CPU implementing a specific instruction set architecture (ISA) together with other standardized hardware. While the ISA specifies the supported CPU instruction and their encoding, the platform allows other computer hardware to be expected to behave in documented ways and specifies, e.g., how the boot process works. As the IBM PC is the almost only x86-based platform, any x86 emulator will actually support it and, thus, can boot and run Android-x86. However, there is not yet any real equivalent for the ARM architecture (this is currently being fixed in https://www.arm.com/architecture/system-architectures/systemready-certification-program).

Given these challenges emulating ARM-based devices, there are currently two non-proprietary options that we've evaluated regarding ensuring long term access to the applications that were used on them:

1. *Android-x86*

Android-x86 is an open-source project with the goal of porting existing Android versions to the x86(-64)/IBM PC platform. The main advantage of Android-x86 is that it can be used with any x86-capable emulator, such as QEMU and works "out-of-the-box". Android-x86 is a community-based effort and thus there is no guarantee that the project will be continued and maintained long-term. Currently the latest release features Android 9, which was released in 2018. The current upstream Android Release is Android 13, with Android 14 being released this year. This shows that there is quite a discrepancy between the latest official Android release and that of Android-x86, however, with a long-term preservation view, this can be disregarded. Android-x86 allows to install the "Native Bridge" feature which allows to install and execute Applications within Android-x86 that were originally compiled to run on ARM-architecture only.

Most users of Android do not run the x86(-64) version.[2] Instead, they run a version of Android compiled for and compatible with ARM(64)-based hardware. ARM hardware has traditionally been more power-efficient than x86 hardware and so has been the default option for mobile devices that have limited battery capacity. Upon initial examination this might be expected to cause significant issues for preserving access to mobile applications as it would be reasonable to assume that most mobile applications were made for ARM-based devices. However Android and mobile applications each have some beneficial features that make this less of an issue that might be expected. From the beginning of mobile development many apps have been designed for either web-browser based execution or for use with a Java Virtual Machine (JVM)[3]. The web-browser based apps often work on any version of Android as they only rely on the in-built web browser. The Java-based apps will run on any version of Android as their architecture-independent bytecode is transparently compiled to machine code on the respective device itself.

2. *Google Android Emulator*

Google provides an official Android Emulator as part of Android Studio, the official Development

[1] Historically, this has often been the case for CPU architectures, considering that the MOS Technology 6502 CPU and its variants were used in such diverse computers (platforms) as the Commodore C64, Apple II, TRS-80, BBC Micro, Super Nintendo, and many others.

[2] However, Google is recently providing an x86-based virtual Android environment on most of their Chromebooks.

[3] Implemented as Android Runtime (ART) on Android (starting with Android 5.0) and being one of core parts of the Android operating system.

Environment for Android. This emulator can also be used in standalone mode and consists of a QEMU with additional features and a somewhat complex emulator architecture [2]. Google provides Android system images, featuring different sizes, resolution, and most importantly Android versions. Images are provided in both x86 and ARM versions and range from Android 1.5 to Android 13. The Google Android Emulator offers a variety of input methods to simulate "real" input that a user would provide to a handheld device. It also contains interfaces to specify locations, use phone services such as simulating incoming calls or messages etc.

From a user standpoint, the Google Emulator seems like the obvious choice to emulate Android especially regarding input. However, there are strong arguments against the usage of the Google Emulator for long-term preservation:

1. As mentioned above, though being built on QEMU, the Google Emulator cannot easily be integrated with the EaaS framework.

2. Maintainability cannot be guaranteed as Google often discards projects and not all of the emulator's functionality is publicly documented well

3. The Google Emulator mainly uses the Android Debug Bridge (ADB) and wraps commands in its interface. If the EaaSI platform is extended to support ADB anyway, the better solution is to use Android-x86 and the "normal" QEMU and build interfaces for ADB functionality as we see need.

2. User Experience

3. Application Installation Workflows

1. The Standard Application Installation Workflow

The usual "workflow" when installing an application is the following: The user opens the Google Play Store, selects the Application that they want to install, and clicks "Install". While this option currently works within emulated Android-x86, it requires both Internet connectivity (which can be provided by Emulation as a Service (EaaS)) and a Google Account. Regarding long-term preservation however, we cannot assume a functional App Store and thus cannot rely on the Google Play Store. Additionally, for security reasons, Google enforces a policy where apps need to target recent Android versions or, otherwise, will not be available through the Play Store.[4] Android Packages (or "APK"s), however, can easily be installed manually, either from within the OS, or externally via a remotely executed command, using ADB.

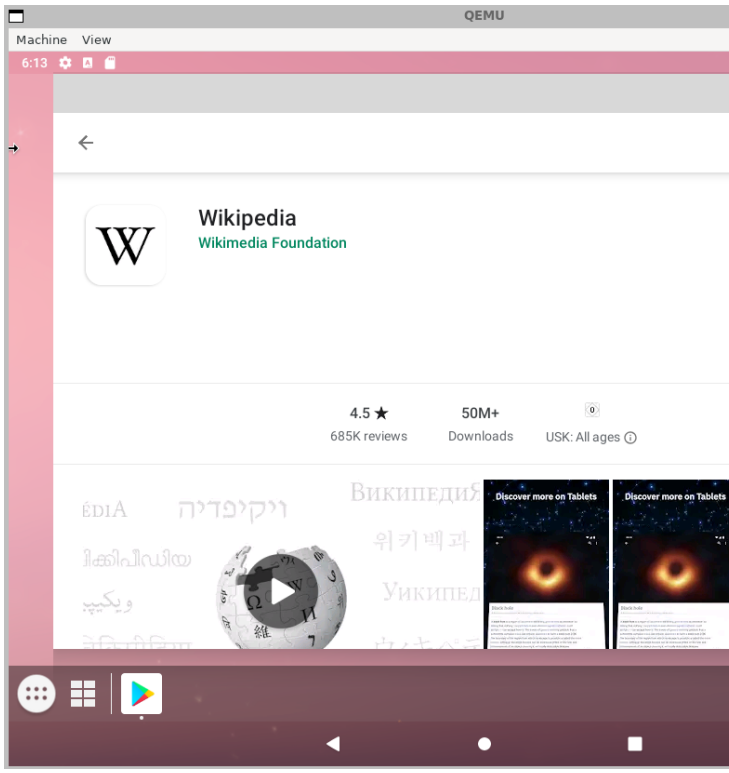

Figure 1: Installing the Wikipedia App within an emulated Android 9

2. Side Loading Applications

By default, and in contrast to Apple's iOS mobile device Operating System (OS), the Android Operating Systems have allowed "side-loading" of applications once a user enabled the relevant system-setting. "Side-loading" is the process of installing an application from a file accessible to the device (e.g., downloaded from the internet, copied on removable media, or accessed from a network location). Apple does not allow this by default as they state that they consider it a security risk [4]. Side-loading in iOS is possible if the operating system and device are "jail broken". However, the process of "jail breaking" a device has many copyright and security concerns associated with it, this is one of a number of reasons why the EaaSI team has begun working with Android

[4] https://android-developers.googleblog.com/2017/12/improving-app-security-and-performance.html

instead of iOS to address long term mobile application preservation and access.

The ability to side-load applications in Android OSes provides a simple and future-proofed method for preservation practitioners to use to ensure preserved mobile applications can be installed and accessed by future users. It is by taking advantage of this option that EaaSI users can use the existing software installation workflow to install applications in Android-based mobile devices emulated in EaaSI. EaaS currently supports installation through the following workflow: The user uploads that APK that they want to install. The backend wraps the APK in an ISO file that can be inserted into the emulated Android system, similar to how an SD card is inserted into a real smartphone. The user can then install the APK from the Android File Browser.

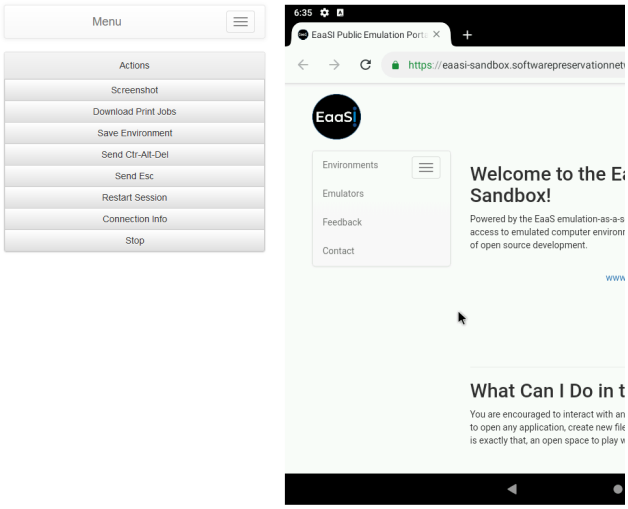

Figure 2: The Google Chrome Browser running within Android 9 in the EaaS UI

3. *Providing a Custom Legacy App Store*

Most users install applications in their mobile Operating Systems (OSes) by finding them in the OS's application store ("app store") and clicking the button to install the application. One option for managing workflows for installing software in preserved and emulated versions of mobile devices and their OSes would be to replicate this process, i.e., the EaaSI software could provide a server hosting the applications and an "app store" application on the emulated devices that would provide a similar experience to the app stores users are used to. This option is relatively complex and requires a network to be setup between the app-store host server and emulated device along with sufficient metadata to be populated into the app store database to enable meaningful searching and browsing within it. Such a configuration would also need to be maintained for as long as the need to install legacy applications was required. For these reasons, while the EaaSI team may explore this option in the future, we have instead decided to start with simpler workflows for enabling applications to be installed on emulated mobile devices.

Automatically Installing Applications

In addition to side-loading applications via the ISO-wrapping workflow described above, the EaaSI team have prepared Android-x86 images that automate this process using a startup script that checks if any APKs are present within the mounted ISO and installs them via the integrated 'package' command line tool. That means that no user input is required, and the installed app(s) can be used shortly after startup. This automated installation is possible because APKs usually don't require any external dependencies and can "just" be installed.

There are currently plans to expand the EaaS-Framework to allow external installation of APKs as well. As networks are already an established functionality in EaaS, a container with ADB could be connected to the Android Emulator. ADB could then be used to remotely install APKs over the network.

Interaction

Interacting with Android applications (or applications for handheld devices in general) differs from interaction with a computer. This poses new challenges for emulation as well. While many applications can run properly in an emulated environment, input and output in the form of multi-touch is not supported in EaaSI yet. While the Google Emulator provides an interface to simulate Inputs, the current EaaSI-UI does not support any of these Android-related features. The above-mentioned network-based ADB integration will set a base for the integration of some of these features, where a UI input would be translated by the backend to an ADB command – which is similar to the way the emulator provided in Google's Android Software Development Kit (SDK) functions.

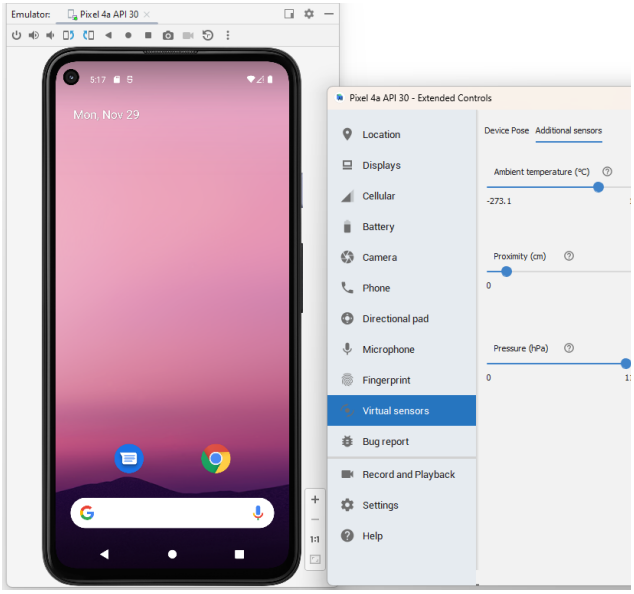

Figure 3: An example of the Google Android Emulator with extended Controls to simulate sensor input

5. *Virtualizing ARM devices*

While we have made great progress with virtualization of x86-compatible mobile devices, virtualization of ARM devices will require some additional work. Currently all servers that run EaaSI are x86(-64)-based and to virtualize ARM devices we would need to incorporate ARM(64)-based servers alongside the x86-based ones and develop EaaSI to support seamlessly utilizing and interacting with each type depending on the device being virtualized. ARM emulation on x86-based hardware will be essential in the future and is already possible. Given the delay between creation of born-digital archives and their acquisition by archival institutions (often many years), it may not ever be necessary to virtualize ARM-based devices in EaaSI as the existing emulation functionality may be performant enough on modern hardware to negate the need to virtualize the ARM devices.

6. *Apple device emulation and virtualization*

As discussed, we are not yet integrating any Apple device emulation or virtualization into EaaSI. There are some existing companies providing Apple device virtualization support, and some that claim to support Apple device emulation. However, in both cases their products are proprietary, which makes integrating them into the fully open-source EaaSI platform particularly challenging. In addition, there has also been significant litigation by Apple against these companies. The Software Preservation Network supported one company, Corellium (a company that supports security testing, training and research), in one such lawsuit by providing an amicus brief [5]. Fortunately, Corellium had some success with that case, however the risk of litigation is still high, which provides motivation for the EaaSI team to delay additional investment into integrating Apple device emulation and virtualization.

2. CONCLUSION

The EaaSI team have made significant progress in ensuring long-term access to mobile applications. A large proportion of historic Android-compatible applications can already be made accessible in EaaSI using the QEMU emulator and the Android-x86 operating system. Automated installation of these applications is also possible in EaaSI for many versions of Android. In the future, the process for installing and configuring these applications will be further simplified and streamlined using the Android Device Bridge. Additionally, configuration of and interaction with the more complex inputs and outputs that mobile devices and their emulators support is going to be integrated into the EaaSI User Interface enabling simple replication of the experience of interacting with legacy mobile devices and their applications.

Challenges remaining to be resolved include emulation and virtualization of Apple's mobile devices, however progress coming from the security testing industry seems to show promise for providing solutions that we may be able to integrate into EaaSI in the future.

3. REFERENCES

[1] Android-x86, Online. https://www.android-x86.org/
[2] https://source.android.com/docs/setup/create/avd, https://android.googlesource.com/platform/external/qemu/+/refs/heads/emu-master-dev
[3] Android Open Source Project, Online. https://source.android.com/
[4] Clover, J. 26/5/2022, MacRumours.com "Apple Says Revised U.S. Sideloading Bill Would 'Undermine the Privacy and Security Protections' iPhone Users Rely On". https://www.macrumors.com/2022/05/26/apple-statement-revised-sideloading-bill/. Accessed 3/9/2023
[5] Butler, Brandon. (2022, February 17). SPN amicus brief defends fair use in Apple v. Corellium case. Software Preservation Network. https://www.softwarepreservationnetwork.org/spn-amicus-brief-defends-fair-use-in-apple-corellium-case/