DebianRunner: Running Desktop Applications on Android Smartphones

Project Report

Abhishek Gupta, Alejandro Rodriguez and Kurt Preston
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
Email: {gupta59, arodri30, kpresto3}@illinois.edu

Abstract

The Android operating system runs atop the Linux kernel, yet lacks the ability to natively execute standard Linux desktop applications. Though efforts have been made to enable this functionality, all approaches to date require a user to switch to a separate desktop environment, resulting in a substantial loss of interface consistency and usability.

This paper describes DebianRunner, an Android application designed to execute in a custom, root-enabled Android build. It unlocks desktop Linux capabilities by mounting the full Debian Linux distribution into the file system, and provides a tool to simplify the installation and execution of Linux programs from within Android.

Though functional, DebianRunner suffered from poor performance as a result of limited RAM in the HTC Dream used for testing. However, more powerful phones should be better able to exploit DebianRunner's functionality, and the most of techniques explored in this paper would be applicable to other rooted Android phones.

1 Introduction

1.1 Background

Since Android's release in October 2008, the operating system has attracted much developer attention, currently supporting over 25,000 publicly released applications [4]. Android has proved a powerful, though insulating programming environment; even though Android runs on top of the Linux kernel and Java runtime environment, the platform prohibits applications from interacting directly with the kernel or Java VM [26]. While these restrictions help present an integrated and secure environment, they also trammel the power of the underlying technologies.

Developers are forbidden from directly accessing hardware (preventing features like tethering), compiling to assembly (limiting performance optimizations), or utilizing the many libraries supported by Linux (preventing things like the execution of Blackberry apps). Furthermore, users are prevented from accessing the rich ecosystem of software already developed for the Linux kernel. The Ubuntu public repositories alone contain over 27,000 packages [22], constituting hundreds of millions of lines of code, refined by tens of thousands of developers over multiple decades, none of which can be directly used in Android [21]. To be used, these applications have to go through the complicated process of porting, often needing to be completely re-written, despite the fact that the technology required to run them exists natively on the device.

1.2 Previous Approaches

Attempts have been made to overcome these shortcomings. To address the performance limitations inherent to Java programs, Google has provided the NDK [3], a toolkit to enable developers to run optimized C/C++ code for performance gains. To simplify the porting of embedded applications, MapuSoft has released a code porting utility [20]. Most notably, some custom Android builds have enabled system calls [27], and have been utilized to install and run alternate desktop environments in parallel with Android [12]. All these approaches are useful, and address some of concerns itemized above; however, none provide users with a simple way to seamlessly integrate external applications.

Although Linux applications can now be run from within a parallel desktop environment, the utility of this feature is lost by its inconvenience: users must switch to an entirely different environment to install or run software. These desktop environments were created for a completely different form factor than a cell phone, and translate poorly onto such a small screen, forcing users to pan and zoom through an oversized desktop to interact with the software. No approaches to date enable non-Android software to behave like a normal part of the system.

1.3 DebianRunner

A number of hardware and software barriers must be overcome to support the execution of non-Android software. Importantly, a customized operating system is required that enables system calls. Furthermore, Android provides no built-in capability for communicating with external processes, requiring all incoming data to be transmitted over network connections using protocols like VNC. Multiple nodes of integration must be addressed: Linux applications should be installed and removed from within Android; the user should be not forced to interact with a different desktop environment when using external applications; windows should be reformatted to fit the smaller display. Ideally, the user is prevented from accidentally corrupting their install. All this must also be done with a minimal memory footprint due to the limited RAM available on smartphones.

With all this in mind, we have developed DebianRunner, an Android application running on top of a custom Android environment that enables improved integration of externally-executed applications into the Android system. Linux desktop applications are launched and interacted with from within an Android application, without requiring a user to switch to a foreign desktop environment. Selecting an application within DebianRunner causes the application to launch in the background, and start streaming graphic data over a remote VNC connection. Though the user is actually operating on a VNC client, the impression of working directly on an application is maintained.

This approach enables broad flexibility, enabling the presentation of any application that can be invoked from system calls. Furthermore, it hides the abstractions from the user, enabling them to treat the external application more like a part of their native environment. In effect, it dramatically expands the number of applications that an Android user can run.

Though currently hindered by the memory limitations of our test hardware, our implementation demonstrates the execution of simple graphical Linux applications. Applications such as gCalculator and gEdit can operate without any modification, and with the larger RAM capacity of modern phones, more complicated applications should be possible.

Much work is still required to provide the user a completely transparent experience; however, we believe that DebianRunner represents an important step forward in the use of Android phones as a platform to execute the thousands of useful legacy applications originally intended for other environments.

The rest of the paper is organized as follows: Section 2 describes the design decisions we have made and the alternatives we have considered; Section 3 describes the results we obtained; Section 4 discusses limitations and potential extensions of our work; Section 5 discusses related work; Section 6 concludes.

2 Design and Implementation

2.1 Design Overview

The integration of Linux desktop applications into Android requires work across a variety of areas: custom Android firmware must be built, system calls must be wrapped, Linux display configuration modified, and a number of user interaction particularities must be addressed. However, our implementation can be divided into four key areas:

1. Enable Android applications to execute privileged Linux commands: Custom Android firmware was used to allow Android applications to make system calls with root privileges (discussed in section 2.3).
2. Restore full Linux capabilities: A Debian Linux image was mounted to provide functionality that was stripped from Androids Linux install (discussed in section 2.4).
3. Provide a mechanism for graphically interacting with applications: Linux's display settings were optimized for the small screen, and graphics were streamed via VNC (discussed in section 2.5).
4. Integrate all user-facing components into an Android application: The DebianRunner application was created to allow a user to run and install Linux applications from within Android (discussed in section 2.6).

These components are all deployed on the same system, as illustrated in figure 1.

To execute applications, DebianRunner performs the following main steps (indicated by the numbers in figure 1):

1. A user selects an application to launch or install from the DebianRunner Android application.
2. A JNI call is made through the NDK, invoking a script to install or run the application.

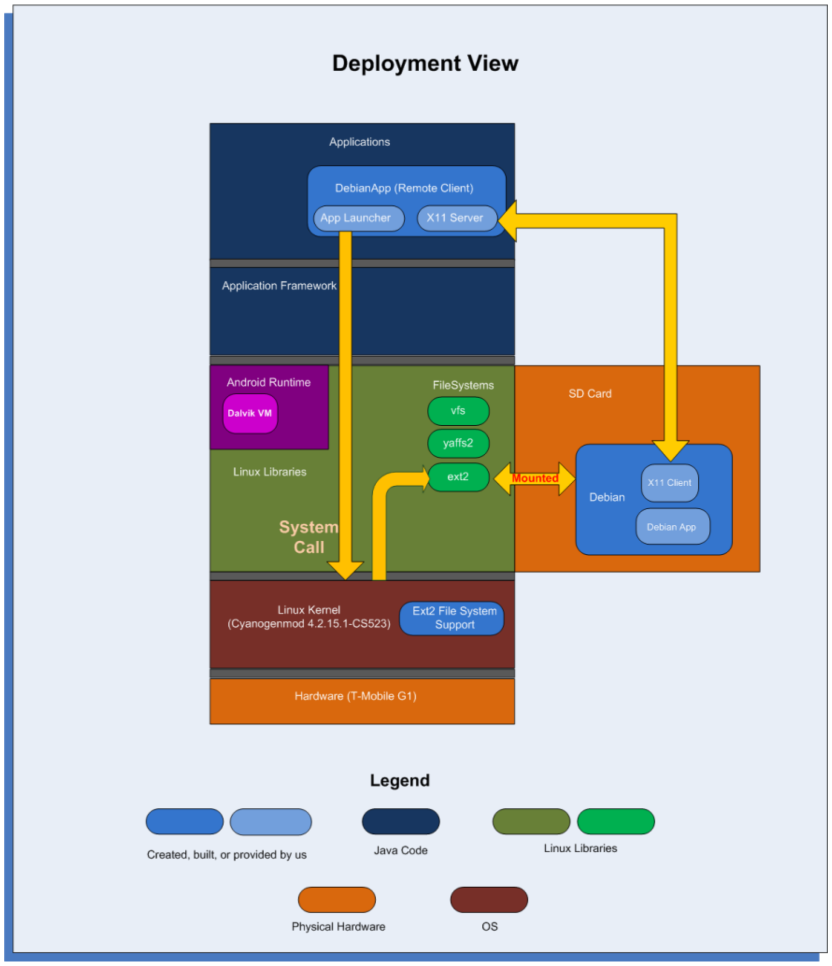


Figure 1: Deployment View

3. The script will execute the desired command within Debian's context (rather than Android's Linux).
4. A VNC server is started in Debian, formatted to fit the phone's screen.
5. The application is launched in Debian.
6. A VNC client is started by DebianRunner, and the Debian application's graphics are streamed to the user.

The technical details of this process are described in further detail in the following sections.

2.2 Target Platform: HTC Dream ("T-Mobile G1")

Although the Android operating system is relatively unified project, there remain notable differences between release versions, and a variety of functional inconsistencies in the firmware. Simply put, Android is a little bit different from phone to phone, version to version, and consequently, a more focused approach was required.

We attempted modifications to the Motorola Droid, as well as configuration of system images loaded by the Android SDK's emulator, but ultimately decided to perform our implementation on the HTC Dream (branded in the United States as the "T-Mobile G2"). Though it provides the most flexibility, the emulated environment proved extremely slow in booting custom images, and seemed to have enough inconsistencies with phone's native capabilities that our implementation might be difficult to port to actual hardware. For example, the emulator by default lacks some extremely basic system utilities such as the Linux "cd" and "pwd" commands, requiring a modification of the image to perform even the most rudimentary tasks. We also explored development on the Motorola Droid, but documentation was vastly inferior to the

Dream/G1.

Due to the volume of available documentation on the Dream/G1 by the Android hacking community, the size of the phone's user base, and our ability to acquire the phone for testing purposes, our implementation was targeted specifically for that phone. However, it is our hope that our methodology could ultimately be generalized, enabling any phone to run Linux desktop applications.

2.3 CyanogenMod: Enabling System calls

Although it runs on top of Linux, Android is designed to be an insular environment [26]. Applications run within a custom Java VM called "Dalvik", and are strictly controlled in their access of system resources [6]. Execution of native system code is limited to controlled libraries compiled using the Android Native Development Kit (NDK) [3]. However, the NDK forbids access to resources outside of Android.

Launching Linux applications thus requires a modification of the Android operating system, and its underlying firmware. This itself is problematic, as most Android phones restrict the user in modifying these system files [9]. However, a security hole was found in the RC29 Android release enabling users to start a telnet daemon as root. By exploiting this vulnerability, users could override the phone's protection mechanisms and flash a custom system image [14]. Custom images were created that enabled users to execute system calls within Android applications, thus potentially enabling users to execute Linux applications.

Our implementation is built atop one of these custom Android images called CyanogenMod [7] since it currently maintains the largest user base of all root-enabled custom Android builds.

2.4 Mounting Debian: restoring full Linux capabilities

By default, Android is deployed on top of a heavily stripped-down Linux install, lacking a desktop environment and many basic Linux commands [15]. Graphical Linux applications require a variety of dependencies which must be installed in order to operate. These required packages were loaded by means of installing and mounting an unpacked image of the Debian Linux distribution.

Running Debian on Android has been done before on various firmware versions of the G1 since late 2008. The most popular method was to mount a Debian image file onto Android's Linux file system as a loopback device. A user by the name Ghostwalker from the androidfanatic.com forum, created a set of scripts that automated the installation and mounting process of Debian onto the G1. However, this only worked on the RC33 firmware version of the G1. Several different instructions for different firmware versions were posted throughout Internet forums in an attempt to describe a procedure that successfully worked on the latest CyanogenMod build. Using as a reference the directions posted by user ChrisDos from the xda-developers.com forum, we were able to successfully install and mount Debian onto a G1 phone having the CyanogenMod 4.2.15.1 firmware version.

Since the 4.2.1.15 firmware had issues mounting loopback devices, we circumvented this limitation by copying the contents of the Debian image onto a 2 GB ext2 partition of the sd card. We modified Ghostwalker's scripts to take into account this workaround, and copied them to the */sdcard/debian/* directory of the sd card's FAT32 partition. Running the *bootdeb* script with the command **su </sdcard/debian/bootdeb** mounted the Debian ext2 partition onto Android's Linux file system, and allowed us to *chroot* into the Debian file system. We were then able to configure the Debian environment, install updates, and most importantly, install and configure the VNC server *tightvncserver*.

2.5 VNC and Devil's Pie: viewing and operating Linux applications

Android provides no built-in capability for visually communicating with external processes, requiring all incoming data to be transmitted over network connections. We explored two protocols to provide this functionalityVNC and X11 [2]. Though our initial effort focused on implementing an X11 client in Android, we ultimately decided to use VNC due to the availability of an existing VNC client for Android [5], as well as a Linux utility called Devil's Pie [10] that enabled applications to be more easily reformatted for the phone's display.

The primary functional difference between these two protocols is that X11 streams graphics on an application-by-application basis, whereas VNC presents a users entire desktop. Since DebianRunner attempts to conceal the Linux desktop from users, X11 seemed a natural fit. Moreover, two open-source Java X11 implementations already existed (WeirdX [24]and Escher [19]).

However, Android applications all run maximizedthat is, they take up the entire screen when launched. If Linux applications were to behave like native ones, they too should take up the entire screen. Accomplishing this requires one of two approaches: either the applications window should be resized when streamed over X11, or the application should be maximized when streamed over VNC. Because Devil's Pie provides the capability to remove windows' borders and menu bar upon launch, maximizing the windows was a viable approach; the windows' minimizing and maximizing controls could be concealed, giving the impression of a single, full-screen application. Thus, VNC was selected, assisted by a Devil's Pie process running in the background.

The actual execution of the VNC server is done by a script, */usr/local/bin/vncstart_480x320*, that automatically starts a VNC session at port 5900 using the G1's screen resolution of 480x320. Since this script resides within Debian's filesystem, we created the script, */sdcard/debian/vncstart_480x320*, which calls */usr/local/bin/vncstart_480x320* and allows Android applications to run Debian programs. The */sdcard/debian/vncstart_480x320* script is as follows:

- export USER=root
- export HOME=/root
- chroot /data/local/mnt /usr/local/bin/vncstart_480x320

In addition, the X11's hosts file had to be configured to allow "remote" execution of graphical applications from the terminal. For the VNC client, we used the open source androidVNC project, but had to make minor modifications to the program to better integrate it. In particular, the client had to be modified to support the "portrait" layout used by most Android applications (by default, the client assumed a landscape orientation). Also, some minor modifications had to be made to allow it to be invoked from within DebianRunner.

This approach was largely successful, but does present some minor inconsistencies for the user. For example, since all Linux applications are viewed in a single VNC client instance, separate applications will not appear as separate tasks on Android's task manager. To switch between Linux applications, a user must select the application in DebianRunner. However, it should still be possible to create desktop shortcuts for specific applications by enhancing DebianRunner with Android's shortcut API.

The resulting layout for the Linux calculator application gCalculator is shown in figure 3.

The LXDE taskbar is shown on the bottom so we could monitor CPU use during testing. This taskbar can ultimately be removed, to present a true single-application view. Font size also appears a bit small, as a result of resizing the application to unusual dimensions. This could be counteracted by changing Debian's text appearance options, although each application may require different ideal sizes, complicating this.

2.6 DebianRunner: Tying the pieces together

With all the back-end components set-up and configured, an Android application was required to provide the user a central source to perform all necessary operations. These include:

1. Mounting the Debian image
2. Starting the VNC server and Devil's Pie
3. Installing applications
4. Uninstalling applications
5. Running/viewing applications

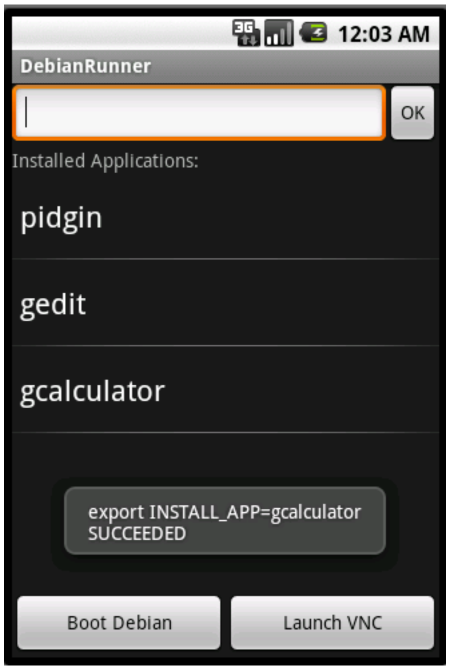
DebianRunner is a simple Android application designed to present a front-end to all these capabilities, and looks as shown in figure 2.

To install applications, a user must simply type the application's name, and press the "OK" button. This will make a system call to execute the install script, passing the application name as a parameter into the apt-get package manager. The application is then added to the list of installed applications. Similarly, applications can be uninstalled by invoking the installed application's context menu, and selecting "uninstall".

Running the application is done by selecting the application from the list of installed applications. Clicking an item invokes the run script, and then launches the VNC client so that the application can be viewed. Two buttons on the bottom execute scripts to set-up the system. "Boot Debian" mounts the Debian partition, and "Launch VNC" starts the VNC server and Devil's Pie.

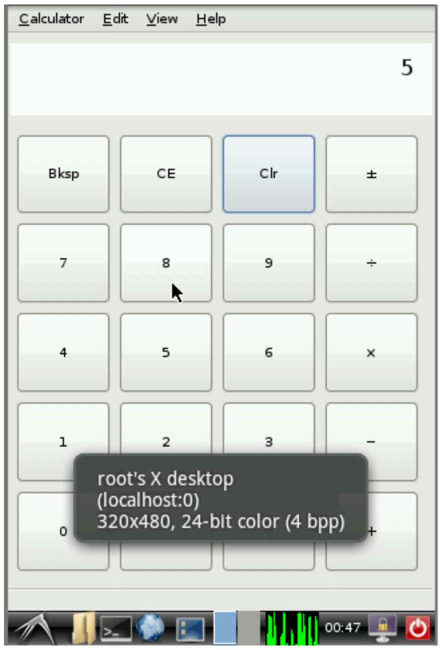
These scripts are executed via the Android NDK, which we used to create a C++ shared library that makes system calls on behalf of DebianRunner, using the JNI interface. This would ultimately allow us to mount Debian, start the VNC server, and install and run Debian programs. The following is the list of scripts we created along with their function:

1. */sdcard/debian/bootdeb* - Mounts Debian fileystem onto Android's Linux filesystem; enables IP4 forwarding
2. */sdcard/debian/vncstart_480x320* - Calls */usr/local/bin/vncstart_480x320*, which resides within Debian's fileystem, to start the VNC server



export INSTALL_APP=gcalculator SUCCEEDED

Figure 2: DebianRunner



root's X desktop (localhost:0) 320x480, 24-bit color (4 bpp)

Figure 3: Layout for the Linux calculator application: gCalculator

3. */sdcard/debian/install_app* - Installs a Debian application

4. */sdcard/debian/run_app* - Runs a Debian application

5. */sdcard/debian/uninstall_app* - Uninstalls a Debian application

DebianRunner currently only provides basic functionality. For example, it does not track running applications, nor does it provide a mechanism for terminating them. Furthermore, the mouse cursor is currently controlled by the trackball, rather than the touch screen, which is abnormal behavior for an Android application. These shortcomings, and other potential extensions to DebianRunner are discussed in section 4.

3 Results

In its current state, DebianRunner has been demonstrated as functional, but suffers from performance and feature limitations. It succeeds in its primary goal of enabling applications to be installed and run from within Android, and provides a solid foundation for future work in the integration of desktop applications. However, the test hardware (HTC Dream) is not sufficiently powerful to provide a truly usable experience.

The primary issue is that of memory. The HTC Dream offers only 192 MB of RAM [16], with 96 MB reserved for non-system purposes (GPU, radio firmware, etc.) [13]. That leaves Android only 96 MB of space to work with, almost all of which is used by Android itself. In our tests, a fresh reboot of Android leaves only 2.3MB to run Debian and its applications. Once Debian is launched, Android frees some memory to make room for the new processes, but there remains only 1.6 MB unclaimed. Android is effective in dynamically adjusting to the amount of memory it has available, but its performance declines as its available RAM is reduced. Given that a comparable Debian install consumed approximately 105 MB on our desktop tests, it is a bit surprising that it even ran at all on our test hardware.

As a rough benchmark, gCalculator takes approximately 0.5s to respond to clicks, and approximately 5 minutes to install. Many larger programs could not even install due to lack of memory. For example, the instant messaging client Pidgin failed to complete installation in our tests –the system simply became unresponsive. Because memory was such a bottleneck, we didnt perform any formal performance benchmarks; the numbers would come out radically differently on a device with more memory. In short, we believe that the lackluster performance DebianRunner isn't a software limitation, but a hardware one.

That said, the hardware specifications of Android phones are increasingly rapidly. For example, the Nexus One phone, released only 15 months after the HTC Dream, comes with 512 MB of RAM –320 MB more than the HTC Dream. This provides more than enough space to hold Debian's 105 MB footprint. Furthermore, Nexus One's CPU is clocked at almost twice the speed (1 GHz, compared to Dream's 528 MHz) allowing for further speed increases.

4 Potential Extensions

Though our implementation was functional, there remain a number of enhancements we did not have the opportunity to implement. These can be divided into two primary categories: opportunities for improved integration, and supplemental features.

4.1 Integration Enhancements

There remain a number of areas in which Debian applications could be more fully integrated with the Android user experience. Below are listed some of the key areas for improvement we perceive:

- **Touch screen support:** Currently, the VNC client does not support touchpad usage for the mouse cursor. The code exists in the client, but is not active due to the way the client is invoked. This can be fixed by enhancements to the VNC client launch procedure.

- **Sound support:** Linux system sounds are not currently streamed to Android. This can be solved by configuring Debian to use a streaming audio server like IceCast [17] and adding code to DebianRunner to receive and play the audio stream.

- **Desktop shortcuts:** Standard Android applications can always be launched by clicking on icon on the desktop or in the "programs" menu. Though it is not currently done, shortcuts to Debian applications could be created on the desktop by utilizing Android's CREATE_SHORTCUT intent; however, icons cannot be added to the "programs" menu without creating a separate Android application (which Android programs lack the permissions to do).

4.2 Feature Enhancements

In addition to integration improvements, there remain a number of capabilities which can be built off of DebianRunner, but are not currently implemented.

- **Windows emulation:** Linux has relatively robust Windows emulation support, notably in the Wine project [25]. If scripts were created to execute custom Wine calls, then Windows applications could be launched much in the same way Linux applications currently are. This would dramatically expand the scope of programs supported by the Android platform to encompass much of the worlds most used software.

- **BlackBerry/Symbian emulation:** Together, BlackBerry and Symbian constitute 42% of the current smartphone market [1]. Emulation could be extended to either of these platforms, utilizing gnupoc [23] in the case of Symbian, and using Wine to run BlackBerry simulators.

- **Task listing and termination:** DebianRunner does not currently provide a mechanism for killing tasks. If launched program ids were to be tracked by DebianRunner, it could conceivably send "kill" commands in response to a user request to quit. Without this feature, DebianRunner runs the risk of consuming too much of the systems memory with long-term use.

5 Related Work

As discussed in section 1, much effort has been made to modify and expand the capability of the Android operating system, but no work was found that attempts to tie non-Android applications directly into the Android environment.

A technology consultant, Jay Freeman (publishing under the name "saurik"), wrote the first well-known article named Debian and Android Together on G1 [8] in late 2008 for installing and running Debian onto the G1. But his method relied on a security hole in the RC 29 firmware version where key-presses were being routed to a Linux console that was running a root terminal. This method became obsolete once the RC30 version patched the bug.

In January 2009, a user by the name Ghostwalker from the `http://www.androidfanatic.com` website created instructions for installing Debian on the G1 using a custom firmware version created by a user named JesusFreke from Android-Roms google project [18]. However, Ghostwalker's instructions only work with this specific firmware version, and are not compatible with other Android-build communities such as Cyanogenmod.

Installation of Debian on the Droid was apparently accomplished a few months ago by a user named Fresh-Meat [11]. However, this required a customized version of the ext2.ko kernel module that is no longer provided by this user.

These efforts have served as a foundation for our research, but have different points of focus. We hope to expand upon their work by the creation of an Android application and custom environment capable of running standard Linux desktop applications, without requiring a user to ever exit Android itself.

6 Conclusion

We believe that DebianRunner represents a significant step forward in the execution of desktop applications on Android smartphones. Though our test hardware suffered from poor performance, we anticipate that these issues will disappear as phones become more powerful, perhaps even on the current generation of Android devices. DebianRunner has the potential to dramatically expand the number of applications executable on the Android platform, encompassing most applications that Linux can run or emulate.

We are optimistic that these capabilities will expand as hardware becomes more sophisticated. Ultimately, Android smartphones phones will have sufficient hardware to run sophisticated spreadsheet applications or image manipulation programs. DebianRunner unlocks some of Android phones' significant hidden capabilities, bringing us one step closer to pocket Starcraft.

References

[1] Admob Mobile Metrics Report. `http://metrics.admob.com/wp-content/uploads/2010/03/AdMob-Mobile-Metrics-Feb-10.pdf`.

[2] An Introduction to X by the Linux Information Project (LINFO). `http://www.linfo.org/x.html`.

[3] Android 1.6 Ndk, Release 1. `http://developer.android.com/sdk/ndk/1.6_r1/index.html`.

[4] Android Market Statistics. `http://www.androlib.com/appstats.aspx`.

[5] Android-vnc-viewer. `http://code.google.com/p/android-vnc-viewer/`.

[6] Application Fundamentals — Android Developers. `http://developer.android.com/guide/topics/fundamentals.html`.

[7] CyanogenMod Android Rom. `http://www.cyanogenmod.com/`.

[8] Debian and Android Together on G1. `http://www.saurik.com/id/10`.

[9] Developing on a Device — Android Developers. `http://developer.android.com/guide/developing/device.html`.

[10] Devil's Pie. `http://burtonini.com/blog/computers/devilspie/`.

[11] Fully working Debian on your stock 2.0.1 Droid. `http://alldroid.org/threads/14749-Fully-working-Debian-on-your-stock-2.0.1-Droid!`

[12] Gnome, Kde, IceWM or LXDE on your Android! `http://www.androidfanatic.com/cms/community-forums.html?func=view&catid=9`.

[13] How much memory should we actually be seeing ? `http://forum.xda-developers.com/showthread.php?t=613205`.

[14] How to Root, Hack and Flash your G1. `http://forum.xda-developers.com/showthread.php?t=442480`.

[15] HOWTO: Unpack, Edit, and Re-Pack Boot Images. `http://android-dls.com/wiki/index.php?title=HOWTO:_Unpack,_Edit,_and_Re-Pack_Boot_Images`.

[16] HTC - Products - HTC Dream - Specification. `http://www.htc.com/www/product/dream/specification.html`.

[17] Icecast.org. `http://www.icecast.org/`.

[18] Installing the Latest Custom ROM. `http://code.google.com/p/android-roms/wiki/Install_Custom_ROM`.

[19] Java X11 Library. `http://sourceforge.net/projects/escher/`.

[20] Mapusoft Extends Support to Android, Enabling Legacy Code Reuse. `http://www10.edacafe.com/nbc/articles/view_article.php?section=ICNews&articleid=779297`.

[21] Sloccount Web for Debian Lenny. `http://libresoft.es/debian-counting/lenny/index.php?menu=Statistics`.

[22] Software Packages in karmic. `http://packages.ubuntu.com/karmic/allpackages`.

[23] Symbian development on Linux and OS X . `http://www.martin.st/symbian/`.

[24] WeirdX - Pure Java X Window System Server under GPL. http://www.jcraft.com/weirdx/.

[25] WineHQ - Run Windows applications on Linux, BSD, Solaris and Mac OS X. http://www.winehq.org/.

[26] L. Batyuk, A.-D. Schmidt, H.-G. Schmidt, A. Camtepe, and S. Albayrak. Developing and Benchmarking Native Linux Applications on Android. In *MobileWireless Middleware, Operating Systems, and Applications*, pages 381–392, 2009.

[27] A.-D. Schmidt, H.-G. Schmidt, L. Batyuk, J. H. Clausen, S. A. Camtepe, S. Albayrak, and C. Yildizli. Smartphone Malware Evolution Revisited: Android Next Target? In *Proceedings of the 4th International Conference on Malicious and Unwanted Software (Malware 2009)*, pages 3–9. IEEE, 2009.

A Set-Up Instructions

1. Create an ext2 filesystem partition on the phone's SD card. Step-by-step instructions can be found at `http://androidandme.com/2009/08/news/how-to-manually-partition-your-sd-card-for-android-apps2sd/`. Using an 8 GB sdhc card with 4 partitions, here is a suggested strategy:

   Partition 1: 5 GB FAT32 - For the /sdcard mount

   Partition 2: 512 MB Ext2 - For the apps2sd

   Partition 3: 512 MB Swap - For improving phone response & usability

   Partition 4: 2 GB Ext2 - For Debian

2. Follow the instructions for downloading the debian image file and installation scripts at `http://www.androidfanatic.com/community-forums.html?func=view\&catid=9\&id=2248`. Make sure to just download the files and not follow the other steps listed since they're out-of-date.

3. Connect and mount the phone onto a Linux system. Make note of the block device that the Debian ext2 partition represents. For example, */dev/sdb4*.

4. Copy the debian image file, *debian.img*, to the */tmp* directory of the Linux system.

5. Execute these commands:

   mkdir /tmp/debloop

   mkdir /tmp/debpart

   mount -o loop /tmp/debian.img /tmp/debloop

   mount /dev/sdb4 /tmp/debpart (Substitute /dev/sdb4 for the actual block device if necessary)

   rsync -av progress

   /tmp/debloop/* /tmp/debpart

   unmount /tmp/debloop

   unmount /tmp/debpart

6. Create the */sdcard/debian* directory on the phone, and copy the downloaded files from step 2 to this directory.

7. Update the *installer.sh* file as shown below, and execute this script using the command **su </sdcard/debian/installer.sh**. *dev/block/mtdblock3* is the block device mounted by the */system* filesystem. If the phone is not a G1, this block device may be different. For example, *dev/block/mtdblock4* is the block device of */system* for Motorola Droid phones. This can be checked using the **df** command.

   # New installer.sh

   mount -o rw,remount -t yaffs2 /dev/block/mtdblock3;

   echo "Filesytem remounted as read/write";

   rm /system/bin/fsrw;

   rm /system/bin/bootdeb;

   rm /system/bin/unionfs;

   rm /system/bin/installer.sh;

   rm /system/bin/mountonly;

   echo "Removed old Installation Files"

   sleep 1;

   mkdir /data/local/mnt;

   echo "Made /data/local/mnt";

   sleep 1;

   cp /sdcard/debian/fsrw /system/bin;

   cp /sdcard/debian/bootdeb /system/bin;

   cp /sdcard/debian/unionfs /system/bin;

   cp /sdcard/debian/installer.sh /system/bin;

   cp /sdcard/debian/mountonly /system/bin;

   echo "Copied new Installation Files";

   echo "";

   echo "VERSION 2.2";

```
echo “Custom Debian Bootloader is now installed!”;
echo “This process does NO damage to your Android OS!”;
echo “”;
echo “Courtesy of http://www.myhangoutonline.com”;
echo “Installer by WhiteMonster84”;
echo “”;
echo “To enter the Debian Linux console just type ‘sh bootdeb’”;
echo “PS: Be sure to run ‘sh /scripts/onetime.sh’ as root from the shell after your FIRST ‘boot’.”;
```

8. Update the *bootdeb* script as shown below, and execute this script using the command **su </sdcard/debian/bootdeb**. Change the */deb/block/mtdblock3* block device to the correct one if needed (described in step 6). Also, change the */dev/block/mmcblk0p4* block device to that of the phone’s actual ext2 Debian partition if needed. Finally, use echo statements between commands if issues are encountered.

```
# Based on Saurik’s remount.sh - modified by WhiteMonster84 of http://www.myhangoutonline.com
# Email whitemonster84@gmail.com
mount -o rw,remount -t yaffs2 /dev/block/mtdblock3;
echo “Android Filesystem remounted as read/write”
export bin=/system/bin;
export mnt=/data/local/mnt;
export PATH=$bin:/usr/bin:/usr/sbin:/bin:$PATH;
export TERM=linux;
export HOME=/root;
mount /dev/block/mmcblk0p4 $mnt;
mount -t devpts devpts $mnt/dev/pts;
mount -t proc proc $mnt/proc;
mount -t sysfs sysfs $mnt/sys;
echo “Custom Linux Pseudo Bootstrapper V2.2. - by WhiteMonster84”
echo “WEB: http://www.myhangoutonline.com”
sleep 1
sysctl -w net.ipv4.ip_forward=1;
```

9. Execute the command **chroot /data/local/mnt /bin/bash** to transfer to the Debian filesystem context.

10. Execute **/scripts/onetime.sh** and choose a password.

11. Modify */etc/apt/sources.list* file to the following:

```
deb urlhttp://ftp.us.debian.org/debian lenny main contrib non-free
deb urlhttp://security.debian.org/lenny/updates main contrib non-free
```

12. Modify */root/.bashrc* to the following:

```
# /.bashrc: executed by bash(1) for non-login shells.
export PATH=“/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin: /bin”
export SHELL=/bin/bash
hostname yourhostname
export PS1=‘\h:\w\$ ’
umask 022
# You may uncomment the following lines if you want ‘ls’ to be colorized:
export LS_OPTIONS=‘–color=auto’
eval “‘dircolors’”
alias ls=‘ls $LS_OPTIONS’
alias ll=‘ls $LS_OPTIONS -l’
alias l=‘ls $LS_OPTIONS -lA’
export USER=root
```

13. Run ssh with the command **/etc/init.d/ssh** start, look up your ip address with **ifconfig**, and ssh into your phone (Windows users can use putty)

14. Update the phone and install some apps with the commands below. Be patient as each command may take a few hours.

```
apt-get update
apt-get-u upgrade
apt-get install vim lxde tightvncserver locales tsclient pptp-linux
```

15. Setup a vnc password by running the command **/root/.vnc/passwd**

16. Create the file */etc/X0.hosts* and add the following lines:

```
local
localhost
```

17. Create the script */usr/local/bin/vncstart_480x320* with the contents shown below, and run it. This will start a VNC session :0. Test it by downloading a VNC viewer from the Android Marketplace and connecting to the VNC server. The IP Address should be *localhost* and the port number should be 5900.

```
#!/bin/bash
vncserver -kill :0
sleep 2
sshagent_pid=‘ps -ef |grep ssh-agent |grep -v grep |awk ‘print $3”
kill -9 $sshagent_pid
sleep 2
xstartup_pid=‘ps -ef |grep xstartup grep -v grep |awk ‘print $2”
kill -9 $xstartup_pid
sleep 2
Xtightvnc_pid=‘ps -ef |grep Xtightvnc |grep -v grep |awk ‘print $2”
kill -9 $Xtightvnc_pid
sleep 2
rm -fr /tmp
mkdir /tmp
chmod 777 /tmp
vncserver :0 -geometry 480x320
```

18. Exit out of the Debian filesystem context, create the script */sdcard/debian/vncstart_480x320* with the contents below, and run it using the command **su </sdcard/debian/vncstart_480x320**. This allows the VNC server to be started within the context of the Android Linux environment as opposed to the Debian environment.

```
export USER=root
export HOME=/root
chroot /data/local/mnt /usr/local/bin/vncstart_480x320
```

19. Create the script */sdcard/debian/install_app* with the contents below, and execute the command **export INSTALL_APP=devilspie; su </sdcard/debian/install_app**. Confirm that the devilspie program was successfully installed on the Debian partition. #!/system/xbin/bash

```
export USER=root
export HOME=/root
echo “Program:”
echo $0
echo $1
echo $2
echo $INSTALL_APP
chroot /data/local/mnt /usr/bin/apt-get install $INSTALL_APP y
```

20. Create the script */sdcard/debian/run_app* with the contents below, and execute the command **export RUN_APP=devilspie; su </sdcard/debian/run_app**. Confirm that the devilspie program successfully started by **executing ps -ef |grep devilspie**.

```
#!/system/xbin/bash
export USER=root
export HOME=/root
export DISPLAY=:0
echo “Run Program: ”
echo $RUN_APP
chroot /data/local/mnt /usr/bin/$RUN_APP &
```

21. Create the script */sdcard/debian/uninstall_app* with the contents below, and execute the command **export UNINSTALL_APP=devilspie; su </sdcard/debian/run_app**. Confirm that the *devilspie* program was uninstalled from the Debian partition.

```
#!/system/xbin/bash
export USER=root
export HOME=/root
echo “Uninstall Program: ”
echo $UNINSTALL_APP
chroot /data/local/mnt /usr/bin/apt-get remove $UNINSTALL_APP y
```

22. Reinstall *devilspie*, create the directory */root/.devilspie*, and create the file */root/.devilspie/remoteall.ds* with the following contents:

```
; generated_rule Some Test
( begin
( undecorate )
( maximize )
( shade )
( println "match" )
)
)
```

23. Add the line *devilspie* to the */usr/local/bin/vncstart_480x320* script to run *devilspie* as part of the VNC server startup.

24. Using the Android NDK, create a C++ shared library that can make system calls to run the */sdcard/debian/install_app*, */sdcard/debian/uninstall_app*, and */sdcard/debian/uninstall_app* scripts.

25. Create a regular Android application that can make native function calls to the C++ shared library via JNI, in order run the scripts created in the previous steps. These include:

    - **su </sdcard/debian/bootdeb** (For mounting the Debian partition onto Android's filesystem)
    - **su </sdcard/debian/vncstart_480x320** (For starting the VNC server)
    - **export INSTALL_APP=devilspie; su </sdcard/debian/install_app** (For installing an application)
    - **export RUN_APP=devilspie; su </sdcard/debian/run_app** (For running as application)
    - **export UNINSTALLINPP=devilspie; su </sdcard/debian/run_app** (For uninstalling an application)