David Krovich
SENG691B
Research Paper
Fall 2016

An Attempt At Porting Ubuntu Phone to a Samsung Galaxy S4

Ubuntu Phone is software developed for mobile devices by Canonical, the company behind the popular Ubuntu Linux Operating system. Ubuntu Phone is a relatively new effort and currently runs on a very limited set of hardware. However, the software behind Ubuntu Phone is open source and porting to alternate platforms is encouraged. This research paper will explore some of the internals of the Ubuntu Phone Operating system and will discuss a port to a Samsung Galaxy S4 phone.

Before going into details related to porting, first we will take a look at the Ubuntu Phone Operating system itself. The Ubuntu Linux operating system serves as the basis for Ubuntu Phone. In order to be mobile friendly, Ubuntu Phone developers constructed a new user interface that runs on top of Ubuntu. This new user interface supports touch screens and is designed to work with smaller mobile screens. This interface is referred to as Ubuntu Touch.

Another feature set being touted with the Ubuntu Phone operating system is Canonical's convergence strategy. This is a strategy by Canonical to use one mobile interface across a wide variety of devices including phones, tablets, and tv set top devices. In addition to providing a common interface using convergence a user can attach their phone via HDMI to a TV and use a bluetooth mouse and keyboard. The Ubuntu phone operating system will detect that it is attached to an external display and readjust the desktop automatically to fit the new screen dimensions. The idea behind this strategy is that you could carry your phone around with you all day and then when you come home dock the device and use it in a desktop mode. All you applications will be installed and all files in place since it is the same device you have been carrying around with you, just in a different mode of operation.

The last feature of Ubuntu phone we will discuss is what is called Scopes. Scopes are an attempt to reinvent the way that users typically experience content on a device. There are two types of Scopes, Aggregation Scopes and Branded Scopes. Aggregation Scopes, as the name implies, will take information from various sources and attempt to bring them together into a unified experience. An example of an aggregation scope could be a music scope that pulls content from a variety of providers. Branded scopes are similar to aggregation scopes but will focus on content from a particular brand or entity. An example of a branded scope is would be an Amazon scope that has content across video and music scopes.

Even though it has a mobile interface, Ubuntu Phone is still linux. The userland is the same as the desktop version. Applications such as Firefox, Libreoffice, The Gimp, etc, can all be executed on Ubuntu Phone. This feature is potentially appealing to developers as they can focus on writing one application that can run on different devices such as phones, tablets, desktops, and tv set top boxes.

In order to support a wide variety of devices Ubuntu Phone uses drivers and hardware enablement available for Android. Ubuntu Phone uses the android linux kernel and runs all basic services needed for Ubuntu Phone in a lxc container. Ubuntu Phone uses the following components from Android. The first is the stock android kernel with a few customizations to add support for additional needed functionality such as AppArmor. The second thing is the OpenGL ES2.0 HAL Drivers. The third thing is Media (stagefright) HAL, to reuse the hardware video decoders. The fourth thing is RILD for modem support. The fifth thing is Android Camera HAL and Camera service. Ubuntu Phone then runs as the main host on top of this android kernel. Communication between the Android services and HAL happens via Binder, Sockets and libhybris. As stated earlier, the Ubuntu Phone component uses the same ubuntu that is used for ubuntu linux.

Before a port can be attempted a developer must understand the toolchain and process used to build the Ubuntu Phone operating system. It is recommended to first learn on a supported architecture before attempting to tackle the port itself. Another thing to consider the fact that Ubuntu relies on Android in order to run. Due to that fact, in order to port Ubuntu Phone to a new device means that for all intents and purposes Android must run on that device as well. With that in mind one strategy for porting is to first focus on building and making Android run before diving into the port of Ubuntu Phone itself. With those points in mind we will then explore compiling Android from source for a Nexus 4 phone. The Nexus 4 phone is a good choice as it is well supported by both Android and Ubuntu Phone. Additionally the phone is a few years old and while still powerful enough to be useful due to age they can be purchased online for relatively cheap.

Before starting to build from source it can be a good idea to first learn to load prebuilt Android images onto a phone. This will help ensure that you have correctly identified the correct hardware information associated with your device. The Nexus 4 has freely available prebuilt Android images available to download. The Nexus 4 supports fastboot, which is the name of both a tool and a protocol that allows for transferring of rom images to a phone. The process for installing prebuilt android images is to first download them from the web. Next, place the phone into fastboot mode by pressing the correct combination of keys. For the Nexus 4 this is achieved by pressing the volume down and power buttons at the same time. Once the phone is in fastboot

mode using fastboot software and a USB cable the downloaded prebuilt Android versions can be installed. Note the phone's bootloader must be unlocked for this process to work. If the process of loading Android from prebuilt images is successful this is one way to verify that you will be building source for the correct hardware platform.

Source code for Android is available from the Android Open Source Project (AOSP). According to the AOSP web site, AOSP is "to refer to the people, the processes, and the source code that make up Android." AOSP is spearheaded by Google. Most of the Android source code is open source software, the majority being licensed under the Apache Software License 2.0. This license was selected due to its permissiveness and flexibility with respect to how source code can be combined and distributed.

The first step in attempting to build Android from source is identifying the exact version that is needed. Android has been existence for many years now and has produced many different versions. Cup Cake, Donut, Eclair, Froyo, Gingerbread, Honeycomb, Ice Cream Sandwich, Jellybean, Kitkat, Lollipop, Marshmallow, and Nougat are all different versions of the Android operating system that has been released over the years. Once that version of Android has been selected this determines the build environment that is necessary to build that version of Android. For Android versions 1.5 (Cupcake) through 2.2.x (Froyo) Ubuntu 10.04 is the reference platform and depends on the Java JDK 5. Android versions 2.3.x (Gingerbread) through 5.x (Lollipop) use Ubuntu 12.04. For Java, Gingerbread through KitKat uses Java JDK 6. Lollipop uses Java OpenJDK 7. Marshmallow uses Ubuntu 14.04 and Java OpenJDK 7. The latest supported version of Android for the Nexus 4 is Lollipop. In order to build images appropriate for this version a combination of Ubuntu 12.04 with OpenJDK version 7 needs to be configured. Additionally, the following packages need to be installed: apt-get install git gnupg flex bison gperf build-essential zip curl libc6-dev libncurses5-dev:i386 x11proto-core-dev libx11-dev:i386 libreadline6-dev:i386 libgl1-mesa-glx:i386 libgl1-mesa-dev g++-multilib mingw32 tofrodos python-markdown libxml2-utils xsltproc zlib1g-dev:i386.

With the build environment in place the next step is to download the correct version of the source. In order to do this a tool named repo needs to be installed. Repo is a tool that was built on top of git to meet the needs of the Android project. Once repo is installed the next step is to initialize repo to work with the android source tree. If you do not want to build the latest version you must specify the specific branch you want with the -b flag to the repo init command. For the Nexus 4 phone being discussed in this paper the branch is android-5.1.1_r19 and the build is LMY48T. Once the repo has been initialized the next step is to actually pull the android source using the repo sync command. The source can be pretty large. In the case of Lollipop 45GB of data was transferred with the repo sync command.

After the source code is downloaded there is one more step that needs to place before Android can be built. This is to download any proprietary hardware drivers that are needed for the particular device you are attempting to build. These drivers will support things such as hardware graphics acceleration, NFC, WiFi, cameras, and other specialized hardware that may be in a device. These drivers are made available by the vendor. For the Nexus 4, 3 binaries need to be downloaded. The first is a driver for NFC Hardware Component that is made by Broadcom. The second is a driver for Camera, Sensors, Audio, DRM, and Cryptography made by LG. The third is a driver for Graphics, GSM, Camera, GPS, Wi-Fi, Bluetooth, Sensors, Media, DRM, DSP, USB made by Qualcomm. The binaries are downloaded in the form of a self extracting script. Before extracting, the user must agree to the terms of a license agreement. If the users accepts, the drivers will then be installed in the vendor/ directory of the Android source tree. The next step is to finalize the environment by using the source command to source build/envsetup.sh script. The envsetup.sh script is used to set various environment variables and setings that are used during compilation. Next the lunch command is invoked and the desired configuration is selected. For the Nexus 4 the configuration to be used is called "aosp_mako-userdebug." After all the setup, Android is finally ready to be built. The process is invoked with the make command. The -j option can be used to the make command to specify the number of threads used to compile. This can help improve the speed of the compilation.

After compilation if all goes well the process will leave a bunch of image files that are suitable for loading onto a phone. Those files are boot.img, cache.img, recovery.img, system.img, and userdata.img. The next part of the process is installing these files onto a phone. For the Nexus phone and many other Android devices this can be achieved via the fastboot protocol. First the phone is placed into fastboot mode usually be pressing a combination of keys. For the Nexus 4 this is achieved by holding down the volume down button while the power is pressed. This will boot android into a maintenance mode where it can talk fastboot. The phone then needs connected to a computer via USB cable. The computer needs to have fastboot software installed in order to communicate via the fastboot protocol. Additionally, the phone's bootloader needs to unlocked in order to load software via fastboot. If all conditions are met, then the images can be installed to the phone over USB cable using fastboot. To do so change to the directory where the images live and issue the command fastboot flashall -w.

With an understanding of the AOSP build and install process the next step is to build Ubuntu Phone, again for the Nexus 4. The Nexus 4 is the reference platform for Ubuntu Phone so it is a good platform to learn on. Similar to when building source from AOSP, the first step is to configure the build environment. The first step is to install the phablet-tools package. Additionally the following packages need to be installed: git, gnupg, flex, bison, gperf, build-essential, zip, bzr, curl, libc6-dev, libncurses5-dev:i386, x11proto-core-dev, libx11-dev:i386, libreadline6-dev:i386, libgl1-mesa-glx:i386, libgl1-mesa-dev, g++-multilib,

mingw32, tofrodos, python-markdown, libxml2-utils, xsltproc, zlib1g-dev:i386, schedtool, g++-4.8-multilib.

The phablet-tools package installs a tool named phablet which is somewhat similar to the Repo tool used to download source from AOSP. As of this writing the stable version of Ubuntu phone is using a modified version based off of the AOSP 4.4.2 source tree. Due to this compiling Ubuntu Phone and compiling AOSP are very similar.

With the build environment the next step is to download the source. First a directory is created, and then the phablet-dev-bootstrap command is executed from that directory. This command will download the branched AOSP Ubuntu Phone source tree. Next it is recommend to set the USE_CCACHE environment variable to 1 to speed up compilation. Similar to AOSP, the next step is to source the build/envsetup.sh script to configure the environment. The lunch command is run and the aosp_mako-userdebug target is selected for the Nexus 4 phone hardware type. Before make can be run any proprietary drivers that need to be installed need to be placed into the vendor/ subdirectory. Since Ubuntu uses the 4.4.2 source tree it is important to make sure to get the correct drivers for that particular version of AOSP. The final step in the compilation process is to run the make command to build the rom images. Once the roms are built and the Ubuntu filesystem injected, they are then loaded onto the Nexus 4 phone via fastboot. The phone is placed into fastboot mode, hooked up to a computer via USB cable, and fastboot software installed on the host computer using the fastboot protocol to transfer rom images Once the rom images are built it is recommend to use a program called rootstock to flash an Ubuntu image containing the Ubuntu userland into the system.img rom. This is what runs out of the primary container that makes up Ubuntu Phone. It is worth noting that as of this writing Ubuntu Phone was based on the 15.10 version of the Ubuntu Linux operating system. In order to use rootstock first the phone is booted into the newly built recovery.img using fastboot boot out/target/product/mako/recovery.img. This will boot the phone into the Ubuntu recovery mode. Once in recovery mode make sure the /data and /cache directories are mounted. This can be accomplished by running adb shell “mount -a” at the command line. After the partitions are mounted run the rootstock-touch-install script and give it the parameters containing the Ubuntu userland img and the system.img. If all goes well everything will be transferred to the phone and it will reboot into Ubuntu Phone.

With background knowledge of building both AOSP and Ubuntu Phone it is time to shift attention to porting to the Samsung Galaxy S4. Armed with the knowledge that Ubuntu Phone is based off the 4.4.2_r2 version of AOSP it makes sense to focus efforts on building that version for the Samsung Galaxy S4 phone. Before focusing on downloading the source first some discussion about the Samsung Galaxy S4 hardware is necessary. Not all Samsung Galaxy S4 phones are created equal. Among the S4 family there are 18 different variations with slightly

different hardware specifications. For this research paper we will be focusing on the SCH-R970ZKAUSC model of the Samsung Galaxy S4 family. This phone was manufactured for use with US Cellular. All references to a Samsung Galaxy S4 phone from here on out will refer to that model.

Unlike the Nexus 4, the Samsung Galaxy S4 is not ready to be built out of the box from the Ubuntu Phone source. The challenge becomes determining what changes are necessary to enable the source code to build for the S4 target. There are 3 main challenges. The first is getting the correct proprietary drivers. The second is getting the /devices directory setup for the S4 hardware. The third is then configuring Mir and potentially other ubuntu specific components to work on the S4.

To get the proprietary drivers there are 3 potential approaches. The first is to go to Samsung and find the correct hardware drivers for your phone and then download them. There are lot of different drivers available so locating the correct drivers for your specific hardware can be tedious. A second approach would be to download drivers from the web. This has risks associated with and it always best to download from official sources. A 3rd potential option is to extract the proprietary blobs from a running system. This requires some understanding of the current installed android environment and assumes the phone is in a somewhat working state.

The /devices directory is also a challenge. One approach is to construct the /devices directory by hand. While this will provide the most in depth learning about the hardware being ported this also the most tedious. Another approach is to look at existing running phone and try to glean information from the phone that way. A last approach would be to use an existing version such as Cyanogenmod that has support for the S4 and then attempt to port that into the Ubuntu fork of AOSP.

At this point in time this is where the current port has progressed. With the knowledge learned thus far the next steps will be to build CyanogenMod (CM) 11 from source. According to the CyanogenMod web site it has support for the US Cellular version of the Samsung Galaxy S4. If successful building and loading CM11 then the next steps will be to attempt to port the CM11 /devices directory for the S4 in the Ubuntu Phone source tree. Next, the new build target would be added and the source would be compiled. If all went well system roms would be built and they would be then loaded onto the S4 using the odin protocol.

References

https://wiki.ubuntu.com/Touch

https://developer.ubuntu.com/en/phone/devices/porting-new-device/

https://nosemaj.org/build-android-4-3-nexus-4

http://dmitry.gr/index.php?r=06.%20Thoughts&proj=02.%20Android%20M%20on%20Nexus4

http://forum.xda-developers.com/chef-central/android/guide-how-to-setup-ubuntu-16-04-lts-t3363669

http://www.androidpolice.com/2015/09/17/software-updates-a-visual-comparison-of-support-lifetimes-for-ios-vs-nexus-devices/

https://www.cnet.com/news/the-chips-of-samsungs-galaxy-s5-exynos-and-snapdragon/

https://zeqiii.github.io/2016/07/10/flash-nexus-device-with-compiled-aosp/

https://developer.ubuntu.com/en/phone/devices/porting-new-device/

http://wccftech.com/unroot-samsung-galaxy-s4-and-reclaim-warranty-all-models/

https://github.com/AOKP/device_samsung_jflteusc

http://www.phonemore.com/samsung-galaxy-s4/models/125

http://www.samsung.com/us/mobile/cell-phones/SCH-R970ZKAUSC-specs

http://updato.com/how-to/android-samsung-update-software-explained-kies-vs-odin

http://news.tecmint.com/all-you-need-to-know-about-ubuntu-scopes/

https://developer.ubuntu.com/en/phone/scopes/

http://www.androidofficer.com/2016/05/r970tyugpd5-android-501-lollipop-update.html

https://samsung-firmware.org/download/GALAXY-S4/60v8/USC/R970TYUGPD5/R970USCGPD5/

http://wiki.cyanogenmod.org/w/Doc:_porting_intro

http://wiki.cyanogenmod.org/w/Doc:_integrated_kernel_building

https://wiki.cyanogenmod.org/index.php?title=Template:Device_jflteusc&action=edit

https://wiki.cyanogenmod.org/w/Build_for_jflteusc

http://galaxys4root.com/galaxy-s4-stock-firmware/

https://wiki.ubports.com/wiki/UBports-Development-Information

https://github.com/JackpotClavin/Android-Blob-Utility