University Of Thessaly

Master Thesis

Packet Routing in android environment for motion management of multiple heterogeneous wireless network interfaces

Δρομολόγηση πακέτων σε περιβάλλον android για τη διαχείριση κίνησης από πολλαπλές ετερογενείς διεπαφές ασύρματων δικτύων

Author:
Zoi Vasileiou

Supervisors:
Dr. Athanasios Korakis
Dr. Leandros Tassioulas
Dr. Antonios Argyriou

A thesis submitted in fulfilment of the requirements for the degree of Master Of Science

in the

University Of Thessaly
Computer and Communications Engineering

September 18, 2014

Declaration of Authorship

I, Zoi VASILEIOU, declare that this thesis titled, 'Packet Routing in android environment for motion management of multiple heterogeneous wireless network interfaces' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

UNIVERSITY OF THESSALY

Περίληψη

Computer and Communications Engineering

Master Of Science

Packet Routing in android environment for motion management of multiple heterogeneous wireless network interfaces

by Zoi VASILEIOU

Ο αρχικός στόχος της διπλωματικής ήταν η εξάπλωση της κυκλοφορίας μέσα από πολλές διεπαφές σε μια android εφαρμογή. Για να το πετύχουμε αυτό, πρέπει να αποσυνδέσουμε την ροή της εφαρμογής από τις IP διευθύνσεις σε κάθε διεπαφή. Μπορούμε να πετύχουμε αυτή την αποσύνδεση χρησιμοποιώντας το λογισμικό OpenVSwitch. Γιάυτό, το πρώτο βήμα αφορά την εισαγωγή του OpenVSwitch kernel module και της διεπαφής χρήστη του στο android. Στην διπλωματική υλοποιήσαμε αυτό το πρώτο βήμα. Πιο συγκεκριμένα, κάναμε cross-compile επιτυχώς το OpenVSwitch kernel module για το Nexus 4, Nexus 5, και την x86 πλατφόρμα. Ωστόσο, κάναμε cross-compile κάποια από τα εργαλεία της διεπαφής χρήστη του OpenVSwitch μόνο για το κινητό Nexus 4. Επειδή δεν είχαμε τα επιθυμητά αποτελέσματα με την εισαγωγή του OpenVSwitch στο android, συμπληρώσαμε αυτη την διπλωματική με την υλοποίηση μιας android εφαρμογής. Υλοποίησαμε τον nitos scheduler ως εφαρμογή στα κινητά με λογισμικό android. Ο NITOS scheduler εχει ήδη υλοποιηθεί ως εφαρμογή του ιστού [3]. Το NITOS είναι μια πειραματική διάταξη που οι κόμβοι του βρίσκονται σε ένα εξαόροφο κτίριο. Το NITOS Scheduler είναι ένα εργαλείο υπεύθυνο για τη διαχείρηση των πόρων.της πειραματικής διάταξης. Αποτελείται κυρίως από δύο κομμάτια, μια web-based διεπαφή χρήστη και ένα κομμάτι εξυπηρετητή που αποτελείται από κάποια scripts και μια βάση δεδομένων. Η διεπαφή χρήστη είναι υπεύθυνη για τη καθοδήγηση των χρηστών κατα τη διαδικασία της κρατησης των πόρων, φροντίζοντας να μην κανει μια κράτηση που συμπίπτει με κρατήσεις άλλων χρηστών. Το κομμάτι του εξυπηρετητη διαχειρίζεται τους πόρους, δηλαδή τους ασύρματους κομβους, και τα κανάλια φάσματος, συσχετίζοντας την αρχή κάθε χρονοθυρίδας (30 λεπτά) κάθε πόρου με το αντίστοιχο slice, σύμφωνα με την πληροφορία που είναι αποθηκευμένη στη βάση δεδομένων. Η διεπαφή που καθοδηγεί τον χρήστη κατα τη διαδικασία της κράτησης υλοποιείται σε αυτή τη διπλωματική ως εφαρμογή για κινητά με λογισμικό android. Είναι σημαντικό να πραγματοποιηθεί μια υλοποίηση του NITOS για πλατφόρμα android, γιατι οι χρήστες που κάνουν κράτηση στους κόμβους από το smartphone τους, θα έχουν καλύτερη εμπειρία από τη διεπαφή, και θα κάνουν ταχύτερες κρατήσεις από το να χρησιμοποιούσαν τον web browser του κινητού.

UNIVERSITY OF THESSALY

Abstract

Computer and Communications Engineering

Master Of Science

Packet Routing in android environment for motion management of multiple heterogeneous wireless network interfaces

by Zoi VASILEIOU

The initial goal of this thesis was to spread traffic from one android application over multiple interfaces. To achieve this, we should decouple the application flow from the IP addresses on each interface. We can do this decoupling by using the OpenVSwitch. Thus, the first step should be the porting of the OpenVswitch Kernel module and its user interface to android. We implement this first step in this thesis. In particular, we cross-compiled successfully the OpenVSwitch kernel module for the Nexus 4, Nexus 5, and android-x86 platform. However, we have cross-compiled successfully some of the OpenVSwitch interface tools only for the Nexus 4.

Since, we did not have the desired results of the porting of OpenVSwitch in android kernel, we enhanced this thesis with the implementation of an android application. We have implemented the nitos scheduler as a mobile application in android. The Nitos Scheduler is already implemented as a web application [3]. NITOS is a testbed whose nodes are laying in a six-floor building. NITOS Scheduler is a tool which is responsible for managing the testbed resources. It is mainly consists of two components, a web-based user interface and a server component comprising of some scripts and a database. The user interface is responsible for guiding the user through the reservation process, making sure that he does not make a reservation conflicting with reservations made by other users. The server component manages the resources, namely wireless nodes and spectrum channels, by associating at the beginning of each time slot (30 min) each resource to the corresponding slice, according to the information stored in the database. The user interface guiding the user through the reservation process is implemented in this thesis in a android mobile application. The biggest advantage of making a mobile application is that it can be customized to leverage the device capabilities that enhance the user experience. Thus, it is worthwhile to make an implementation of the NITOS Scheduler in the Android Platform, since the users reserving the resources through their smartphone have better user experience and faster reservations than using the web browser of the mobile phone.

Acknowledgements

I offer my sincerest gratitude to my supervisor, Dr Korakis Athanasios. Also, i express my thanks to my co-supervisors Leandros Tassiulas, and Argyriou Antonios. Furthermore, I am grateful to Aris Dadoukis and Ioannis Igoumenos for their guidance and help. The most special thanks goes to my parents and friends who gave me unconditional support and love through all this long process.

Contents

List of Figures

Abbreviations

AOSP	**A**ndroid **O**pen **S**ource **P**roject
JSON	**J**ava**s**cript **O**bject **N**otation
NITLAB	**N**etwork **I**mplementation **T**estbed **L**aboratory
OVS	**O**pen **VS**witch
UI	**U**ser **I**nterface

This work is dedicated to my parents, teachers and my best friends...

Chapter 1

Introduction

1.1 Porting OpenVSwitch to Android's kernel

Poor connectivity is common when using wireless networks on the go. Connectivity comes and goes, throughput varies, latencies can be extremely unpredictable, and failures are frequent. Industry reports that demand is growing faster than wireless capacity, and the wireless crunch will continue for some time to come. Yet users expect to run increasingly rich and demanding applications on their smart-phones, such as video streaming, anywhere anytime access to their personal files, and online gaming;all of which depend on connectivity to the cloud over un- predictable wireless networks. Given the mismatch between user expectations and wireless network characteristics, users will continue to be frustrated with application performance on their mobile computing devices.

The good news[7] is that smart-phones will be armed with multiple radios capable of connecting to several networks at the same time. Whereas today's phones commonly have four or five radios (e.g. 3G, 4G, WiFi, Bluetooth), in future they will have more. Shrinking geometries and energy- efficient circuit design will lead to mobile devices with more radios/antennas; a mobile device will talk to multiple APs at the same time for improved capacity, coverage and seamless handover.

We need to spread traffic from one application over multiple interfaces. The application sends traffic using one IP source address; We should do this using a virtual Ethernet interface to connect the application, with its local IP address, to a special gateway inside the Linux kernel. The gateway stitches multiple interfaces together, without the application knowing. Essentially, the gateway is a traffic load-balancer that demultiplexes flows using Open vSwitch, with appropriate changes made to the routing table and ARP tables. In this way, the application flow is decoupled from the IP addresses

on each interface, which allows the set of interfaces to change dynamically as connectivity comes and goes. So, we need to port Open vSwitch to android linux kernel so as change dynamically how each flow is routed. Since the cross-compiling of Open VSwitch on android was not successful, i changed my thesis subject implementing an Android application, the NitosTools application. In chapter 2, we describe all the attempts to cross-compile Open VSwitch for the android linux kernel, and in the next chapters we discuss about the NitosTools android application.

1.2 NitosTools Application

Mobile devices have drastically shifted the online landscape to the point that in 2010 more than 50 percent of all Internet access was being done via handhelds of some sort. The most common mobile operating systems (OS) used by modern smart phones include Google's Android, Apple iOS, Microsoft's Windows Phone etc. Google's Android platform has grown tremendously in the latest years. Android is a mobile operating system (OS) based on the Linux kernel and currently developed by Google. The main advantage of Android is that it is an open source operating system so almost anyone can create apps for it. The goal of this thesis is the implementation of the nitos scheduler as a mobile application, while it is implemented as a web application. NITOS is is a testbed whose nodes are laying in a six-floor building. NITOS Scheduler is a tool which is responsible for managing the testbed resources. It is advantageous to make an implementation of the NITOS Scheduler in the Android Platform, since the users, reserving the resources through their smartphone, can have better user experience and faster reservations than using the web browser of the mobile phone. The biggest advantage of making a mobile application is that it can be customized to leverage the device capabilities that enhance the user experience. So, a mobile application can enhance the user experience than a web application.

1.3 Thesis Structure

In Chapter 2 all the efforts for porting OpenVSwitch to android are described in detail. Chapter 3 discusses about the required theoretical background. In Chapter 4 the system design and architecture is described as well as a high level class explanation. Chapter 5 discusses the implementation details and presents the platforms, the software development. In Chapter 6 a user manual of NitosTools is presented. Finally, Chapter 7 presents our conclusions and directions for future work.

Chapter 2

Compiling Open VSwitch for android

The ultimate purpose is to send traffic over multiple interfaces [7]. We need to spread traffic from one application over multiple interfaces. The application sends traffic using one IP source address; the networking stack takes care of spreading the traffic over several interfaces, each with its own IP address. We will achieve this using a virtual Ethernet interface to connect the application, with its local IP address, to a special gateway inside the Linux kernel. The gateway stitches multiple interfaces together, without the application knowing. Essentially, the gateway is a traffic load-balancer that demultiplexes flows using Open vSwitch, with appropriate changes made to the routing table and ARP tables. In this way, the application flow is decoupled from the IP addresses on each interface, which allows the set of interfaces to change dynamically as connectivity comes and goes. So, sending traffic over multiple interfaces is achieved through OpenVSwitch. Thus, the first step should be the porting of the OpenVswitch Kernel module and its user interface to android.

2.1 OpenVswitch

Open vSwitch (OVS) replaces the bridging code in Linux, and lets us dynamically change how each flow is routed [7]. The main purpose of Open vSwitch is to provide a switching stack for hardware virtualization environments, while supporting multiple protocols and standards used in computer networks.

OpenVSwitch Architecture(see Figure 2.1)

The OpenVSwitch User Interface is responsible for adding and deleting interfaces to the kernel module, and adding and deleting forwarding entries:

- ovs-vswitchd:a deamon that implements the switch, along with a companion Linux kernel module for flow-based switching.
- ovsdb-server:a lightweight database server that ovs-vswitchd queries to obtain its configuration.
- ovs-vsctl:a utility for querying and updating the configuration of ovs-vswitchd
- ovs-dpctl:a tool for configuring and monitoring the switch kernel module.
- ovs-appctl:a utility that sends commands to running OpenVSwitch deamons(ovs-vswitchd)
- ovs-controller: a simple OpenFlow controller reference implementation.

OpenVswitch kernel modules:

- brcompat.ko: Linux bridge compatibility module
- openvswitch.ko: works with bridge and forwards the packets according to the forwarding table entries

OpenFlow is added as a feature to commercial Ethernet switches, routers and wireless access points. As mentioned above, the ovs-controller of the OpenVSwitch is a simple OpenFlow implementation. In a classical router or switch, the fast packet forwarding (data path) and the high level routing decisions (control path) occur on the same device. An OpenFlow Switch separates these two functions. The data path portion still resides on the switch, while high-level routing decisions are moved to a separate controller, typically a standard server. The OpenFlow Switch and Controller communicate via the OpenFlow protocol. The main advantage of OpenFlow is that allows you to easily deploy innovative routing and switching protocols in your network. It is used for applications such as virtual machine mobility, high-security networks and next generation ip based mobile networks.

We should run OVS in kernel space, and port it to Android by patching and and cross-compiling its kernel module and user-space control programs using Android Native Development Kit (NDK) for the ARM or OMAP processors.

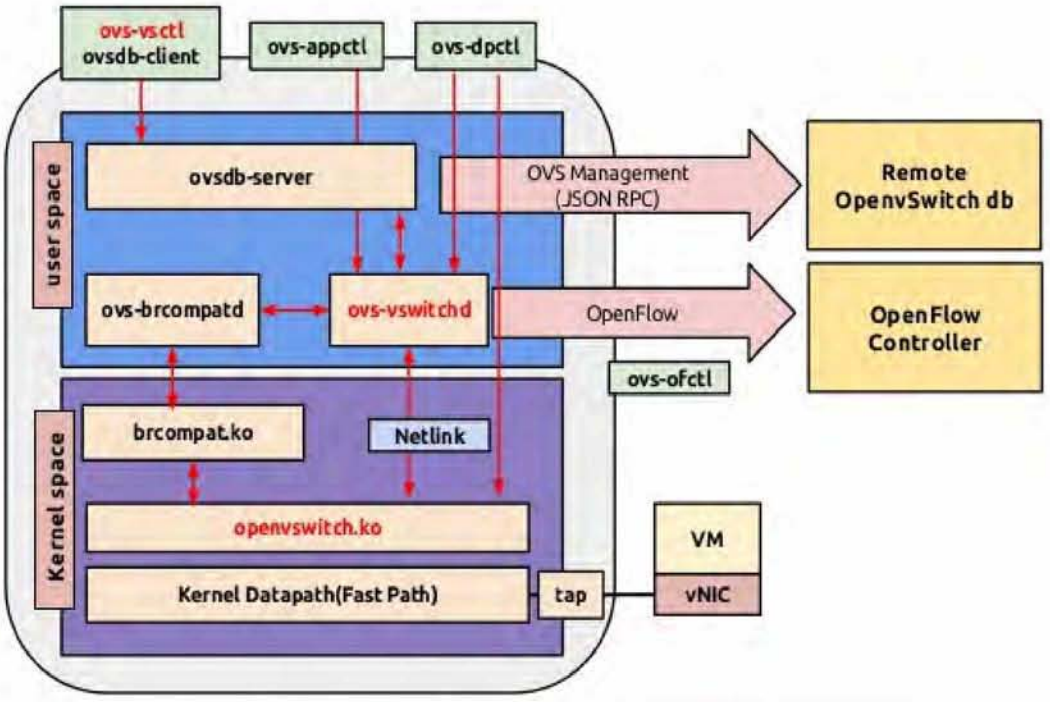

FIGURE 2.1: OpenVSwitch Architecture[12]

To port ovs kernel module in android linux kernel, we should first compile android linux kernel from source and the android source tree(application framework, libraries) and then flash the produced images on a phone. The next step is to cross-compile the openvswitch and its user interface for android, and port them to a phone. At the following sections, all the steps for building android from source and cross compiling the OVS are described. We discuss these steps about four mobile phone and one virtual machine with android-x86 OS. At the first section, we describe some instructions which should be done for all the mobile phones.

2.2 General Instructions

2.2.1 Initializing the Build Environment

In order to set up your local work environment to build the Android source files, we have followed the instruction at AOSP's site[7]. You will need to use Linux or Mac OS. Building under Windows is not currently supported.

Note: The source download is approximately 8.5GB in size. You will need over 30GB free to complete a single build, and up to 100GB (or more) for a full set of builds.

2.2.2 Downloading the source

This step is the same for all the mobile phones to which we have flashed AOSP ROMS. Before download the preferred branch of AOSP, it is necessary to download the repo tool. Repo is a tool that makes it easier to work with Git in the context of Android. To install Repo:

- Make sure you have a bin/ directory in your home directory and that it is included in your path:

```
mkdir ~/bin
PATH=~/bin:\$PATH
```

- Download the Repo tool and ensure that it is executable:

```
curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
chmod a+x ~/bin/repo
```

2.2.3 Downloading the source efficiently

Fortunately, you can instruct repo to download several projects in paralell using the "-j" flag, so the repo sync command will read something like:

```
repo sync -j16
```

-j N where N is the number of CPU cores of the host PC. If the CPU cores are hyper-threaded, N is 2 * number of CPU cores. This option will considerably lower the time needed to download the sources. Althought, be careful, as raising this number too much might be counter productive. Another handy option you might consider using is the "-c" flag: -c, –current-branch fetch only current branch from server That is where the "-c" flag is useful. With that flag, repo will only download the branch that is specified in the manifest, not all the branches that are on the remote server. It will thus save us quite some space, and again it will take less time to download. We downloaded the AOSP tree on a PC with 4 cores using the following command:

```
repo sync -c -j4
```

Troubleshooting network issues

More rarely, Linux clients experience connectivity issues, getting stuck in the middle of downloads (typically during "Receiving objects"). It has been reported that tweaking the settings of the TCP/IP stack and using non-parallel commands can improve the situation. You need root access to modify the TCP setting:

```
sudo sysctl -w net.ipv4.tcp_window_scaling=0
repo sync -j1
```

2.2.4 Flashing the device

After you have downloaded, and built the system, you should flash the images on the mobile phone. We decribe here, the flashing process since it is the same for all the phones. It's only possible to flash a custom system if the bootloader allows it. The bootloader is locked by default. With the device in fastboot mode, the bootloader is unlocked with fastboot oem unlock The procedure must be confirmed on-screen, and deletes the user data for privacy reasons. It only needs to be run once.

There are two ways to boot the device into fastboot mode:

- adb reboot bootloader
- Press and hold both Volume Up and Volume Down, then press and hold Power

Once the device is in fastboot mode, run fastboot flashall -w . The way fastboot knows what files to push onto the phone is via the environment variables set by lunch. It's also very important that you execute fastboot while in the root repo directory(/AOSP/ in our case). Otherwise the tool will hang. The phone should reboot itself automatically. If it does not, use the bootloader menu to do so manually. Your new AOSP build should be on your phone. You'll notice the snazzy colored google X boot screen is now gone. If you go into the "About Phone" options screen, you'll notice the version is set to AOSP and you'll have a different kernel build. Congratulations, you're done.

2.3 Build AOSP for GALAXY NEXUS - maguro and cross-compiling openvswitch

This section describes the procedure for setting up a development environment and then compiling the entire Android Operating System Project (AOSP) from source code for the Galaxy Nexus. This includes all system software, as well as the kernel. Once built,

loading your AOSP build onto your phone involves permanently flashing your device. We will flash a Galaxy Nexus device with android 4.3 and build number JRW66Y. We can take these characteristics from Settings ¿ About phone.

We should cross-compile the android tree and kernel from source and then cross-compile the openvswitch for android linux kernel.

We have used the following versions:

```
Android version:  4.3 (jelly bean)
Android Kernel source git:  https://android.googlesource.com/kernel/msm.git
Kernel version:  3.0
Android Build: full_maguro
Openvswitch version: (Open vSwitch) 1.11.0
```

The directory structure of different source codes is the following(host OS: Ubuntu 13.04):

```
ANDROID_KERNEL: /home/zoe/AOSP_MAGURO/omap
ANDROID_SOURCE_CODE_PATH: /home/zoe/AOSP_MAGURO/
ANDROID_NDK_ROOT: /home/zoe/android-ndk-r4/
OpenVswitch Source: /home/zoe/openvswitch-1.11.0/
ANDROID_SDK: /home/zoe/android-sdk-linux/
```

2.3.1 Building AOSP for NEXUS 4

Downloading the source

- Create an empty directory to hold your working files.Give it any name you like:

```
mkdir WORKING_DIRECTORY
cd WORKING_DIRECTORY
```

- Run repo init to bring down the latest version of Repo with all its most recent bug fixes. You must specify a URL for the manifest, which specifies where the various repositories included in the Android source will be placed within your working directory. To check out a branch specify it with -b:

```
repo init -u https://android.googlesource.com/platform/manifest -b android-4.3_r1.1
```

- When prompted, configure Repo with your real name and email address. To use the Gerrit code-review tool, you will need an email address that is connected with

a registered Google account. Make sure this is a live address at which you can receive messages. A successful initialization will end with a message stating that Repo is initialized in your working directory. Your client directory should now contain a .repo directory where files such as the manifest will be kept.

To pull down the Android source tree to your working directory from the repositories as specified in the default manifest, run:

```
repo sync
```

Obtaining proprietary binaries

Android Open-Source Project can't be used from pure source code only, and requires additional hardware-related proprietary libraries to run, specifically for hardware graphics acceleration. Official binaries for the supported devices can be downloaded from Google's Nexus driver page Google's Nexus driver page which add access to additional hardware capabilities with non-Open-Source code.We downloaded the binaries for Galaxy Nexus(GSM/HSPA+) ("maguro") with build number Android 4.3 (JWR66Y).

Extracting the proprietary binaries

Each set of binaries comes as a self-extracting script in a compressed archive. After uncompressing each archive, run the included self-extracting script from the root of the source tree, confirm that you agree to the terms of the enclosed license agreement, and the binaries and their matching makefiles will get installed in the vendor/ hierarchy of the source tree. [edit] Cleaning up when adding proprietary binaries In order to make sure that the newly installed binaries are properly taken into account after being extracted, the existing output of any previous build needs to be deleted with

```
make clobber
```

Fix for the camera and gps

Camera and GPS don't work on Galaxy Nexus. As an example, the Camera application crashes as soon as it's launched. Those hardware peripherals require proprietary libraries that aren't available in the Android Open Source Project. To fix this we pull the appropriate files from a phone with the factory image. We download the factory images "yakju" for the galaxy nexus "maguro" with build number JWR66Y. We set the phone to fastboot mode, and we flash all the factory images executing the script ./flash-all.sh.

Then we need to pull the appropriate files for fixing the camera and the gps from the phone which now is in factory state. We pull the following files with the adb tool: This is for gps

```
adb pull /system/vendor/etc/sirfgps.conf
```

This is for camera

```
adb pull /system/vendor/firmware/ducati-m3.bin
```

This is for gps

```
adb pull /system/vendor/lib/hw/gps.omap4.so
```

Next, add the following to vendor/samsung/maguro/proprietary/Android.mk

```
include $(CLEAR_VARS)
LOCAL_MODULE := sirfgps
LOCAL_MODULE_OWNER := samsung
LOCAL_SRC_FILES := sirfgps.conf
LOCAL_MODULE_TAGS := optional
LOCAL_MODULE_SUFFIX := .conf
LOCAL_MODULE_CLASS := SHARED\_LIBRARIES
LOCAL_MODULE_PATH := $(TARGET\_OUT\_VENDOR)/etc
include $(BUILD_PREBUILT)
include $(CLEAR_VARS)
LOCAL_MODULE := gps.omap4
LOCAL_MODULE_OWNER := samsung
LOCAL_SRC_FILES := gps.omap4.so
LOCAL_MODULE_TAGS := optional
LOCAL_MODULE_SUFFIX := .so
LOCAL_MODULE_CLASS := SHARED_LIBRARIES
LOCAL_MODULE_PATH := $(TARGET\_OUT\_VENDOR)/lib/hw
include $(BUILD_PREBUILT)
include $(CLEAR_VARS)
LOCAL_MODULE := ducati-m3
LOCAL_MODULE_OWNER := samsung
LOCAL_SRC_FILES := ducati-m3.bin
LOCAL_MODULE_TAGS := optional
LOCAL_MODULE_SUFFIX := .bin
LOCAL_MODULE_CLASS := SHARED\_LIBRARIES
LOCAL_MODULE_PATH := $(TARGET\_OUT\_VENDOR)/firmware
include $(BUILD\_PREBUILT)
```

We found a hint that we should change in the same file the LOCAL_MODULE_PATH for fRom to install into (TARGET_OUT)/bin rather than (TARGET_VENDOR_OUT)/bin. This drastically speeds up the front / back camera switching and stops the video camera from crashing. This hint was unnecessary as the path was already (TARGET_OUT)/bin. Lastly, change your packages list in vendor/samsung/maguro/device-partial.mk to:

```
PRODUCT_PACKAGES :=     \
   fRom         \
   libsec-ril   \
   libsecril-client     \
   sirfgps      \
   ducati-m3    \
   gps.omap4
```

Now we are ready compile the AOSP tree.

Changing Toolchain

AOSP has prebuilt toolchains for cross-compiling the source code. We will describe how it is possible to change the toolchain so as to use our own toolchain. We downloaded the toolchain from the android NDK version r8e. The toolchain used is located in the folder android-ndk-r8e/toolchains/arm-linux-androideabi-4.7. Copy unziped prebuilt toolchain folder to working-folder/prebuilt/gcc/linux-x86/arm.

Note: We will not use the AOSP prebuilt kernel image, but we will produce our own kernel image cross compiling the kernel source code. We will use the same toolchain for cross-compiling the kernel source code with the previous one. The environment is initialized with the build/envsetup.sh script. We should change some paths so as our toolchain will be used. Open build/envsetup.sh Find this block of code:

```
   export ANDROID_EABI_TOOLCHAIN=
   local ARCH=$(get_build_var TARGET_ARCH)
   case $ARCH in
       x86) toolchaindir=x86/i686-linux-android-$targetgccversion/bin
           ;;
       arm) toolchaindir=arm/android-ndk-r8e/toolchains/arm-linux-androideabi-$targetgccversion/
           ;;
       mips) toolchaindir=mips/mipsel-linux-android-$targetgccversion
       /bin;;
       *)
           echo "Can't find toolchain for unknown architecture: $ARCH"
           toolchaindir=xxxxxxxxx
           ;;
   esac
   if [ -d "$gccprebuiltdir/$toolchaindir" ]; then
       export ANDROID_EABI_TOOLCHAIN=$gccprebuiltdir/$toolchaindir
   fi
unset ARM_EABI_TOOLCHAIN ARM_EABI_TOOLCHAIN_PATH
   case $ARCH in
       arm)
           toolchaindir=arm/android-ndk-r8e/toolchains/arm-linux-androideabi-$targetgccversion/p
           if [ -d "$gccprebuiltdir/$toolchaindir" ]; then
                export ARM_EABI_TOOLCHAIN="$gccprebuiltdir/$toolchaindir"
```

```
                ARM_EABI_TOOLCHAIN_PATH=":$gccprebuiltdir/$toolchaindir"
            fi
            ;;
        mips) toolchaindir=mips/mips-eabi-4.4.3/bin
            ;;
        *)
            # No need to set ARM_EABI_TOOLCHAIN for other ARCHs
            ;;
    esac
```

Replace bold text with your toolchain folder name The file /build/core/combo/target_linux_arm contains a variable TARGET_GCC_VERSION which specifies the version of the toolchain. We changed it to 4.7 as the toolchain's version is 4.7.

Initialize

Now we are ready to initialiaze the environment: Initialize the environment with the envsetup.sh script.

```
. build/envsetup.sh
```

Choose a target

Choose which target to build with lunch. If run with no arguments lunch will prompt you to choose a target from the menu. The exact configuration can be passed as an argument. lunch BUILD-BUILDTYPE All build targets take the form BUILD-BUILDTYPE, where the BUILD is a codename referring to the particular feature combination. the BUILDTYPE is one of the following:

Buildtype Use user limited access; suited for production userdebug like "user" but with root access and debuggability; preferred for debugging eng development configuration with additional debugging tools Userdebug is pretty much the standard. We chose the configuration for the GSM galaxy nexus with the following command:

```
lunch full_maguro-userdebug
```

Build the code

Build everything with make. GNU make can handle parallel tasks with a -jN argument, and it's common to use a number of tasks N that's between 1 and 2 times the number

of hardware threads on the computer being used for the build. Our host pc has 4 cores which are not hyper-threaded, so we can have totally 4 threads. So, the fastest builds are made commands between make -j4 and make -j8. We built it with the following command: make -j4 Assuming everything went well, you should have the following images(our WORKING_DIRECTORY is named as AOSP):

```
~/AOSP/out/target/product/maguro/boot.img
~/AOSP/out/target/product/maguro/system.img
~/AOSP/out/target/product/maguro/recovery.img
~/AOSP/out/target/product/maguro/userdata.img
```

The kernel boot.img was not built from scratch. Rather, boot.img was built using an included pre-compiled kernel located in /AOSP/device/samsung/maguro/kernel folder. At the next section we will describe how to build our own kernel and boot.img. We will flash the device after building our own boot.img.

Building the kernel

As stated previously, the build process until now builds the boot.img based off a pre-compiled kernel image. If you have a specific reason to modify the kernel you should build the kernel from scratch.

Downloading the source

We are going to go to the working directory(our working directory is AOSP), and then fetch the Galaxy nexus kernel source code from omap.

```
cd AOSP
git clone https://android.googlesource.com/kernel/omap.git
```

This will give you a git repository with all actual branches of the omap Kernels. omap is the corresponding kernel source for Maguro devices (Galaxy Nexus) After the download you have a omap folder. Note, this repo has to be updated and managed separately from the core aosp repo in /aosp/. That means you need to run git pull from the /aosp/omap/ directory after running repo sync from /aosp/ when updating the entire system from the master repo. In the folder /AOSP/device/samsung/tuna/ do:

```
git log --maxcount=1 kernel
```

This command produces a commit message. The commit message for the kernel binary contains a partial git log of the kernel sources that were used to build the binary in question. The first entry in the log is the most recent(it is specified with the parameter

–maxcount=1), i.e. the one used to build that kernel. You will need it at a later step. The git log command produces a commit message like this:

```
commit 85ae29d3542fd28315cefe3a433bf867e24b3349 Author: JP Abgrall <j...@google.com> Date: Fri J
   prebuilt kernel (DDK 1.8@2112805 version change)

   3b0c5d2 gpu: pvr: Update to DDK 1.8@2112805

   Change-Id: I4158335a06e588f62340848040aae06db1c36d71

   Conflicts:

       kernel
```

Check out the branch

Then enter to the folder /AOSP/omap:

```
cd omap
```

and checkout the branch:

```
git checkout <commit_from_first_step>
```

For instance, referred to the previous commit message, the ¡commit_from_first_step¿ is 3b0c5d2:

```
git checkout 3b0c5d2
```

After the checkout you have all files available.

Note: If you want to see all the branches in your /AOSP/omap folder type:

git branch -a

and the output will be:

```
* (no branch)
 master
 remotes/origin/HEAD -> origin/master
 remotes/origin/android-omap-3.0
 remotes/origin/android-omap-panda-3.0
 remotes/origin/android-omap-steelhead-3.0-ics-aah
 remotes/origin/android-omap-tuna-3.0
 remotes/origin/android-omap-tuna-3.0-ics-mr1
 remotes/origin/android-omap-tuna-3.0-jb-mr0
 remotes/origin/android-omap-tuna-3.0-jb-mr1
 remotes/origin/android-omap-tuna-3.0-jb-mr1.1
 remotes/origin/android-omap-tuna-3.0-jb-mr2
```

```
remotes/origin/android-omap-tuna-3.0-jb-pre1
remotes/origin/android-omap-tuna-3.0-mr0
remotes/origin/android-omap-tuna-3.0-mr0.1
remotes/origin/glass-omap-xrr02
remotes/origin/glass-omap-xrr35
remotes/origin/glass-omap-xrr64b
remotes/origin/glass-omap-xrr88
remotes/origin/glass-omap-xrs36
remotes/origin/glass-omap-xrs68
remotes/origin/glass-omap-xrs92
remotes/origin/glass-omap-xrt35
remotes/origin/glass-omap-xrt73b
remotes/origin/linux-omap-3.0
remotes/origin/master
remotes/origin/sph-l700-fh05
```

This will list you a list of all branches available.Then you can checkout the branch like this:

```
git check out name_of_branch
```

For instance:

```
git checkout android-omap-tuna-3.0-jb-mr1.1
```

Building

Set some environment variables: We built the AOSP tree with an ARM cross compiler of the NDK toolchain. The NDK toolchain is located in the folder /home/user/AOSP/prebuilts/gcc/linux-x86/arm. We use the same cross-compiler for building the kernel. So, we add the path of the cross-compiler in the PATH environment variable: export PATH=/home/user/AOSP/prebuilts/gcc/linux-x86/arm/android-ndk-r8e/toolchains/arm-linux-androideabi-4.7/prebuilt/linux-x86_64/bin:$PATH

```
export ARCH=arm
export SUBARCH=arm
export CROSS_COMPILE=arm-linux-androideabi-
```

Cleaning the Source To ensure that you generate a really new kernel type:

```
make clean
make mrproper
```

This prepares the make configuration:

```
make tuna_defconfig
```

After this we start the make process: make NOTE:Again, we can use the -jx parameter to speed up the make. x can be specified taking into account the number of the supported hardware threads.

When make is finished you should have the message "Creating zImage successfully" The kernel binary is output as: 'arch/arm/boot/zImage'. To flash it, you need to make it into a boot.img.

Create boot.img

What we get out of this process is a zImage. What we want to do is copy that zImage into the android source tree so we can get a boot.img to use on our device. To do this, we copy it to the tuna device folder, so run:

```
cp arch/arm/boot/zImage ../../device/samsung/tuna/kernel
```

The reason we copy zImage to kernel is because that is what the build system looks for. It was already there in the source we downloaded, but we want to use our own kernel that we just compiled. Now, we will convert the zImage to a boot.img. The build/envsetup.sh script, and the lunch command set the environment variables. If you've changed your environment (by closing the terminal or using another window) initialize the source tree again like specified above in the Building the system using source and lunch:

```
. build/envsetup.sh
lunch full_maguro-userdebug
```

Once that is done, we get a boot.img file by running in the root directory /AOSP/:

```
make bootimage
```

When it is done, you'll have a boot.img file in the /AOSP/out/target/product/samsung/maguro/ folder along with the system, userdata, and recovery images that can be flashed to your device via fastboot. Most information for this section was taken from official AOSP page [5].

2.3.2 Cross compiling OpenVswitch

Configure the OpenvSwitch source code:

```
cd openvswitch
./boot.sh
./configure ovs_cv_use_linker_sections=no -with-l26=<android_kernel_dir> KARCH=arm
```

```
--with-rundir=/data/local/var
```

Make change to your makefile datapath/linux-2.6/Makefile.main, add the following line into Makefile: change the $(MAKE) -C $(KSRC) M=$(builddir) modules to $(MAKE) -C $(KSRC) M=$(builddir) ARCH=$(ARCH) CROSS_COMPILE=$(CROSS_COMPILE) modules $ ARCH=arm CROSS_COMPILE=arm–android-eabi- make

Generate Needed Files for Android Apps

```
cd openvswitch
ARCH=arm CROSS\_COMPILE=arm-linux-androideabi- make
```

After running make, find the kernel module under ./datapath/linux-2.6/ with the name openvswitch.ko. Then push the files to the android phone: $ adb push openvswitch_mod.ko /data/local/tmp $ adb shell insmod /data/local/tmp/openvswitch_mod.ko

However, there are errors during the cross-compilation. These errors appear due to the linux kernel 3.072 which may not be compatible with the OpenVSwitch. Changes are needed to become to the OpenVswitch code. One reason is that android does not support any 128 bit value transfer between variables. Therefore the ipv6 addr can not be copied directly in android. Some changes should be done in android linux/ipv6.h and net/ipv6.h header files which have some macro and inline functions to deal with such 128 bit value not being used by the openvswitch. Maybe, other changes are necessary, since the android linux kernel 3.0.72 is still not compatible with the OpenVSwitch. Also, the whole source code of OVS is an overkill to the andriod platform since some libraries may not be supported.

If we got the openvswitch.ko kernel module, the next step would be the cross-compilation the OVS user interface.There are instructions about the steps of this cross-compilation[8].

2.4 Building AOSP for NEXUS 4 (occam (mako) and cross compile OpenVSwitch

We should cross-compile the android tree and kernel from source and then cross-compile the openvswitch for android linux kernel.

We have used the following versions:

```
Android version:  4.2.2 (jelly bean)
Android Kernel source git:  https://android.googlesource.com/kernel/msm.git
Kernel version:  3.4
Android Build: full_mako-usedebug 4.2.2
Openvswitch version: (Open vSwitch) 1.11.0
```

The directory structure of different source codes is the following(host OS: Ubuntu 13.04):

```
ANDROID_KERNEL: /home/zoe/AOSP_MAKO/msm
ANDROID_SOURCE_CODE_PATH: /home/zoe/AOSP_MAKO/
ANDROID_NDK_ROOT: /home/zoe/android-ndk-r9d/
OpenVswitch Source: /home/zoe/openvswitch-1.11.0/
ANDROID_SDK: /home/zoe/android-sdk-linux/
```

2.4.1 Building AOSP for NEXUS 4

Downloading the source

After installing Repo, set up your client to access the Android source repository:

- Create an empty directory to hold your working files. If you're using MacOS, this has to be on a case-sensitive filesystem. Give it any name you like:

```
mkdir WORKING_DIRECTORY
cd WORKING_DIRECTORY
```

- Run repo init to bring down the latest version of Repo with all its most recent bug fixes. You must specify a URL for the manifest, which specifies where the various repositories included in the Android source will be placed within your working directory.

To check out the branch, we do:

```
repo init -u https://android.googlesource.com/platform/manifest -b android-4.2.2_r1.2
```

Then we run

```
repo sync -j4
```

to download the AOSP source code.

Building the system

Choose a target

```
source build/envsetup.sh
lunch full_mako-userdebug
```

Building the code

```
make -j4
```

Building the kernel

Enter in the root directory

```
cd AOSP_MAKO
```

Then download the appropriate kernel branch for the nexus 4:

```
git clone https://android.googlesource.com/kernel/msm.git
git branch -a
cd msm
```

And then we checkout the kernel source code:

```
git checkout android-msm-mako-3.4-jb-mr1
```

We compiled the kernel with the android-ndk-r9d/toolchains/arm-linux-androideabi-4.6 toolchain. The compilation steps follow:

```
export PATH=/home/zoe/android-ndk-r9d/toolchains/arm-linux
-androideabi-4.6/prebuilt/linux-x86_64/bin:\$PATH

export ARCH=arm

export SUBARCH=arm

export CROSS_COMPILE=arm-linux-androideabi-

make mako_defconfig

make
```

The kernel image file zImage will be created in the arch/boot/arm/ folder.

2.4.2 Cross compiling OpenVswitch

Environment setup

```
cd /home/zoe/openvswitch/
```

Executing this script for transfering some files from the android linux kernel to openvswitch source code:

```
./envsetup.sh
```

Apply the ovs patch [11] for changing the openvswitch source code:

```
git apply -v ovs_android.patch
```

For creating the configure file:

```
./boot.sh
```

The contents of the envsetup.sh file is:

```
cp /home/zoe/AOSP_MAKO/msm/include/linux/genetlink.h /home/zoe/android-ndk
-r9d/platforms/android-18/arch-arm/usr/include/linux/
cp ./android_system_files/external/tcpdump/icmp6.h /home/zoe/android-ndk
-r9d/platforms/android-18/arch-arm/usr/include/netinet/
cp ./android_system_files/external/tcpdump/ip6.h /home/zoe/android-ndk
-r9d/platforms/android-18/arch-arm/usr/include/netinet/
cp /home/zoe/AOSP/AOSP_MAKO/msm/include/linux/gen_stats.h /home/zoe/android-ndk
-r9d/platforms/android-18/arch-arm/usr/include/linux/
cp /home/zoe/AOSP_MAKO/msm/include/linux/ethtool.h /home/zoe/android-ndk
-r9d/platforms/android-18/arch-arm/usr/include/linux/
cp /home/zoe/AOSP_MAKO/msm/include/linux/if_tunnel.h /home/zoe/android-ndk
-r9d/platforms/android-18/arch-arm/usr/include/linux/
cp /home/zoe/AOSP_MAKO/msm/include/linux/mii.h /home/zoe/android-ndk
-r9d/platforms/android-18/arch-arm/usr/include/linux/
```

Configure the openvswitch:

```
./configure --with-linux=/home/zoe/AOSP_MAKO/msm KARCH=arm --host=arm-linux-androideabi
 CC=arm-linux-androideabi-gcc CPPFLAGS="-I/home/zoe/android-ndk
 -r9d/platforms/android-18/arch-arm/usr/include/ -I/home/zoe/AOSP/bionic/libc/include/
 " CFLAGS="-D__ARM_ARCH_5__ -D__ARM_ARCH_5T__ -D__ARM_ARCH_5E__ -D__ARM_ARCH_5TE__
 -DANDROID -DSK_RELEASE -DNDEBUG -UDEBUG -march=armv5te  -mtune=xscale -msoft-float
 -mthumb-interwork -fpic -fno-exceptions -ffunction-sections -funwind-tables -fmessage
 -length=0  -mtune=xscale -msoft-float" LDFLAGS="-Bdynamic -Wl,
 -T,/home/zoe/AOSP/prebuilts/gcc/linux-x86/arm/arm-linux-androideabi-4.6/arm-linux
 -androideabi/lib/ldscripts/armelf_linux_eabi.x -Wl,--entry=main,-dynamic
 -linker=/system/bin/linker -Wl,--gc-sections -Wl,-z,nocopyreloc -Wl,--no-undefined
  -Wl,-rpath-link=/home/zoe/android-ndk-r9d/platforms/android-18/arch-arm
   -L/home/zoe/android-ndk-r9d/platforms/android-18/arch-arm/usr/lib
   -L/home/zoe/AOSP/out/target/product/mako/obj/lib -nostdlib /home/zoe/android-nd
   k-r9d/platforms/android-18/arch-arm/usr/lib/crtend_android.o /home/zoe/android-ndk
```

```
   -r9d/platforms/android-18/arch-arm/usr/lib/crtbegin_dynamic.o" LIBS="-lc
    /home/zoe/AOSP/prebuilts/gcc/darwin-x86/arm/arm-linux-androideabi-4.6/lib/gcc/arm
    -linux-androideabi/4.6.x-google/libgcc.a -lm -llog -lgcc -lcrypto -lssl"
```

Compile the source code

```
ARCH=arm CROSS_COMPILE=arm-linux-androideabi- make
```

Successful run creates the library file libopenvswitch.a libovsdb.a libsflow.a. Also It will create the kernel module in /home/zoe/openvswitch/datapath/linux/openvswitch.ko

Now we use following gcc commands to build the different user space program of openvswitch:

```
cd utilities

openvswitch/ovsdb/ovsdb-tool
openvswitch/ovsdb/ovsdb-server
openvswitch/utilities/ovs-ofctl
openvswitch/utilities/ovs-dpctl
openvswitch/vswitchd/ovs-vswitchd
openvswitch/utilities/ovs-vsctl
openvswitch/datapath/linux/openvswitch.ko
openvswitch/vswitchd/vswitch.ovsschema
```

Compile ovs-dpctl utility (It was compiled successfully)

```
/home/zoe/android-ndk-r9d/toolchains/arm-linux-androideabi-4.6/prebuilt/linux
-x86_64/bin/arm-linux-androideabi-gcc -o ovs-dpctl -I//home/zoe/android-ndk
-r9d/platforms/android-8/arch-arm/usr/include -I. -I../ -I../include/ -I../ofproto/
 -I../lib/ -D__ARM_ARCH_5__ -D__ARM_ARCH_5T__ -D__ARM_ARCH_5E__ -D__ARM_ARCH_5TE__
 -DANDROID -DSK_RELEASE -DNDEBUG -UDEBUG -march=armv5te -mtune=xscale
 -msoft-float  -mthumb-interwork -fpic -fno-exceptions
 -ffunction-sections -funwind-tables -fmessage -length=0
 -march=armv5te -mtune=xscale -msoft-float -mthumb-interwork -fpic
 -fno-exceptions -ffunction-sections -funwind-tables -fmessage-length=0
 ovs-dpctl.c -Bdynamic -Wl,-T,/home/zoe/android-ndk-r9d/toolchains/arm-linux
 -androideabi-4.6/prebuilt/linux-x86_64/arm-linux
 -androideabi/lib/ldscripts/armelf_linux_eabi.x -Wl,-dynamic-linker,/system/bin/
 linker -Wl,--gc-sections -Wl,-z,nocopyreloc -Wl,--no-undefined -Wl,
 -rpath-link=/home/zoe/android-ndk-r9d/platforms/android-8/arch-arm
 -L/home/zoe/android-ndk-r9d/platforms/android-8/arch-arm/usr/lib -
 L/home/zoe/AOSP_MAKO/out/target/product/mako/obj/lib
  -nostdlib  /home/zoe/android-ndk-r9d/platforms/android-8/arch
  -arm/usr/lib/crtend_android.o /home/zoe/android-ndk-r9d/platforms
 /android-8/arch-arm/usr/lib/crtbegin_dynamic.o
  -lc /home/zoe/android-ndk-r9d/toolchains/arm-linux-androideabi
  -4.6/  prebuilt/linux-x86_64/lib/gcc/arm-linux-androideabi/4.6/
  libgcc.a /home/zoe/openvswitch-1.11.0/ofproto/libofproto.a
   /home/zoe/openvswitch-1.11.0/lib/libopenvswitch.a
   /home/zoe/openvswitch-1.11.0/lib/libsflow.a -lm
  -ldl -lgcc -lcrypto -lssl
```

cd ../vswitchd (It was compiled successfully)

Compile ovs-vswitchd utility.

```
/home/zoe/android-ndk-r9d/toolchains/arm-linux-androideabi-4.6/
prebuilt/linux-x86_64/bin/arm-linux-androideabi-gcc ovs-vswitchd.c bridge.c
 system-stats.c xenserver.c -o ovs-vswitchd
-I/home/zoe/android-ndk-r9d/platforms/android-8/arch-arm/usr/include
-I. -I../ -I../include
-I../ofproto -I../lib -D__ARM_ARCH_5__ -D__ARM_ARCH_5T__
-D__ARM_ARCH_5E__ -D__ARM_ARCH_5TE__
 -DANDROID -DSK_RELEASE -DNDEBUG -UDEBUG  -march=armv5te
  -mtune=xscale -msoft-float -mthumb -interwork -fpic -fno-exceptions
   -ffunction-sections  -funwind-tables -fmessage-length=0
 -march=armv5te -mtune=xscale -msoft-float -mthumb-interwork
  -fpic -fno-exceptions -ffunction -sections -funwind-tables
  -fmessage-length=0 -Bdynamic -Wl, -T,/home/zoe/android-ndk-r9d/
  toolchains/arm-linux-androideabi-4.6 /prebuilt/linux-x86_64/arm-linux
 -androideabi/lib/ldscripts/armelf_linux_eabi.x -Wl,-dynamic-linker,
 /system/bin/linker -Wl,--gc- sections -Wl,-z,nocopyreloc -Wl,
 --no-undefined - Wl,-rpath-link=/home/zoe/android-ndk
 -r9d/platforms/android-8/arch-arm -L/home/zoe/android-ndk-r9d/
 platforms/platforms/android-8/arch -arm/usr/lib
 -L/home/zoe/AOSP_MAKO/out/target/product/mako/obj/lib/
  -nostdlib /home/zoe/android-ndk-r9d/platforms/android-8/arch
  -arm/usr/lib/crtend_android.o /home/zoe/android-ndk -r9d/platforms/
  android-8/arch-arm/usr/lib/crtbegin_dynamic.o -lc /home/zoe/android-ndk
  -r9d/toolchains/arm-linux-androideabi-4.6/prebuilt/linux-x86_64/lib/gcc/arm-linux
  -androideabi/4.6/libgcc.a /home/zoe/openvswitch-1.11.0/ofproto
 /libofproto.a  /home/zoe/openvswitch-1.11.0/lib/libopenvswitch.a
  /home/zoe/openvswitch-1.11.0/lib/libsflow.a
  -lm -ldl -lgcc -lcrypto -lssl
```

Compile ovs-ofctl utility (It was compiled successfully)

```
/home/zoe/android-ndk-r9d/toolchains/arm-linux-androideabi-4.6/prebuilt/linux
-x86_64/bin/arm-linux-androideabi-gcc -o ovs-ofctl -I//home/zoe/android-ndk
-r9d/platforms/android-8/arch
-arm/usr/include -I. -I../ -I../include/ -I../ofproto/ -I../lib/
-D__ARM_ARCH_5__
-D__ARM_ARCH_5T__ -D__ARM_ARCH_5E__ -D__ARM_ARCH_5TE__ -DANDROID -DSK_RELEASE
-DNDEBUG -UDEBUG
-march=armv5te -mtune=xscale -msoft-float -mthumb-interwork -fpic
-fno-exceptions -ffunction
-sections -funwind-tables -fmessage-length=0 -march=armv5te -mtune=xscale
 -msoft-float -mthumb
-interwork -fpic      -fno-exceptions -ffunction-sections -funwind-tables
-fmessage-length=0 ovs
-ofctl.c -Bdynamic -Wl,-T,/home/zoe/android-ndk-r9d/toolchains/arm-linux
-androideabi-4.6/prebuilt/linux-x86_64/arm-linux-androideabi/lib/ldscripts/
armelf_linux_eabi.x
-Wl,-dynamic-linker,/system/bin/linker -Wl,--gc-sections -Wl,-z,nocopyreloc
```

```
-Wl,--no-undefined
-Wl,-rpath-link=/home/zoe/android-ndk-r9d/platforms/android-8/arch-arm
 -L/home/zoe/android-ndk-r9d/platforms/platforms/android-8/arch-arm/usr/lib
-L/home/zoe/AOSP_MAKO/out/target/product/mako/obj/lib/ -nostdlib
/home/zoe/android-ndk-r9d/platforms/android-8/arch-arm/usr/lib
/crtend_android.o /home/zoe/android-ndk-r9d/platforms/android-8/arch
-arm/usr/lib/crtbegin_dynamic.o  -lc /home/zoe/
android-ndk-r9d/toolchains/arm-linux-androideabi-4.6/prebuilt/linux-x86_64/lib/
gcc/arm-linux-
androideabi/4.6/libgcc.a /home/zoe/openvswitch-1.11.0/ofproto/libofproto.a
 /home/zoe/openvswitch-1.11.0/lib/libopenvswitch.a
  /home/zoe/openvswitch-1.11.0/lib/libsflow.a -lm
  -ldl -lgcc -lcrypto -lssl
```

Compile ovsdb-server utility (It was failed to compile)

```
/home/zoe/android-ndk-r9d/toolchains/arm-linux-androideabi-4.6/prebuilt/linux
-x86_64/bin/arm-linux-androideabi-gcc -o ovsdb-server -I//home/zoe/android-ndk
-r9d/platforms/android-8/arch-arm/usr/include -I. -I../ -I../include/ -I../ofproto/
-I../lib/ -D__ARM_ARCH_5__ -D__ARM_ARCH_5T__ -D__ARM_ARCH_5E__ -D__ARM_ARCH_5TE__
 -DANDROID -DSK_RELEASE -DNDEBUG -UDEBUG -march=armv5te -mtune=xscale -msoft-float
  -mthumb-interwork -fpic -fno-exceptions -ffunction-sections -funwind-tables -fmessage
 -length=0 -march=armv5te -mtune=xscale -msoft-float
 -mthumb-interwork -fpic  -fno-exceptions -ffunction-sections -funwind-tables
 -fmessage-length=0  ovsdb-server.c  -Bdynamic -Wl,-T,/home/
 zoe/android-ndk-r9d/toolchains/arm-linux
 -androideabi-4.6/prebuilt/linux-x86_64/arm-linux
 -androideabi/lib/ldscripts/armelf_linux_eabi.x -Wl,-dynamic-linker,/system/bin
 /linker -Wl,--gc-sections -Wl,-z,nocopyreloc -Wl,--no-undefined -Wl,-rpath
 -link=/home/zoe/android-ndk-r9d/platforms/android-8/arch-arm -L/home/zoe/android-
 ndk-r9d/platforms/platforms/android-8/arch-arm/usr/lib
 -L/home/zoe/AOSP_MAKO/out/target/product/mako/obj/lib/
  -nostdlib /home/zoe /android-ndk-r9d/platforms/android-8/arch
  -arm/usr/lib/crtend_android.o /home/zoe/android-ndk-r9d/platforms
  /android-8/arch-  arm/usr/lib/crtbegin_dynamic.o  -lc /home/zoe/android-ndk-r9d/
  toolchains/arm-linux-androideabi-4.6/prebuilt/ linux-x86_64/lib/gcc/arm
  -linux-androideabi/4.6/libgcc.a
   /home/zoe/openvswitch-1.11.0/ofproto/libofproto.a
  /home/zoe/openvswitch-1.11.0/lib/libopenvswitch.a
   /home/zoe/openvswitch-1.11.0/lib/libsflow.a -lm
   -ldl -lgcc -lcrypto -lssl
```

Compile ovs-vsctl utility (It was compiled successfully)

```
/home/zoe/android-ndk-r9d/toolchains/arm-linux-androideabi-4.6/prebuilt/linux
-x86_64/bin/arm-linux-androideabi-gcc -o  ovs-vsctl.c -I//home/zoe/android-ndk
-r9d/platforms/android-8/arch-arm/usr/include -I. -I../ -I../include/ -I../ofproto/
 -I../lib/ -D__ARM_ARCH_5__ -D__ARM_ARCH_5T__ -D__ARM_ARCH_5E__ -D__ARM_ARCH_5TE__
  -DANDROID -DSK_RELEASE -DNDEBUG -UDEBUG -march=armv5te -mtune=xscale -msoft-float
  -mthumb-interwork -fpic -fno-exceptions -ffunction-sections -funwind-tables
  -fmessage-length=0 -march=armv5te -mtune=xscale -msoft-float -mthumb-interwork
```

```
-fpic      -fno-exceptions -ffunction-sections -funwind-tables
-fmessage-length=0  ovs-vsctl.c -Bdynamic -Wl,-T,/home/zoe/
android-ndk-r9d/toolchains/arm-linux-androideabi-4.6/prebuilt/linux-x86_64/arm
-linux-androideabi/lib/ldscripts/armelf_linux_eabi.x -Wl,-dynamic
-linker,/system/bin/linker -Wl,--gc-sections -Wl,-z,nocopyreloc -Wl,
--no-undefined -Wl,-rpath-link=/home/zoe/android-ndk-r9d/platforms/android-8/arch
-arm -L/home/zoe/android-ndk-r9d/platforms/platforms/android-8/arch-arm/usr/lib
-L/home/zoe/AOSP_MAKO/out/target/product/mako/obj/lib/
-nostdlib /home/zoe/android-ndk-r9d/platforms/android-8/arch
-arm/usr/lib/crtend_android.o /home/zoe/android-ndk-r9d/platforms/android-8
/arch-arm/usr/lib/crtbegin_dynamic.o -lc /home/zoe/android-ndk-r9d/toolchains
/arm-linux-androideabi-4.6/prebuilt/linux-x86_64/lib/gcc/arm-linux
-androideabi/4.6/libgcc.a /home/zoe/openvswitch-1.11.0/ofproto/libofproto.a
/home/zoe/openvswitch-1.11.0/lib/libopenvswitch.a
/home/zoe/openvswitch-1.11.0/lib/libsflow.a -lm -ldl -lgcc -lcrypto -lssl
```

Compile ovsdb-tool utility (It was failed to compile)

```
/home/zoe/android-ndk-r9d/toolchains/arm-linux-androideabi-4.6/prebuilt/linux
-x86_64/bin/arm-linux-androideabi-gcc -o  ovsdb-tool.c -I//home/zoe/android-ndk
-r9d/platforms/android-8/arch-arm/usr/include -I. -I../ -I../include/ -I../ofproto/
-I../lib/ -D__ARM_ARCH_5__ -D__ARM_ARCH_5T__ -D__ARM_ARCH_5E__ -D__ARM_ARCH_5TE__
-DANDROID -DSK_RELEASE -DNDEBUG -UDEBUG -march=armv5te -mtune=xscale -msoft-float
-mthumb-interwork -fpic -fno-exceptions -ffunction-sections -funwind-tables -fmessage
-length=0 -march=armv5te -mtune=xscale -msoft-float -mthumb-interwork -fpic
 -fno-exceptions -ffunction-sections -funwind-tables -fmessage-length=0
  ovsdb-tool.c -Bdynamic -Wl,-T,/home/zoe/android-ndk-r9d/toolchains/arm-linux
  -androideabi-4.6/prebuilt/linux-x86_64/arm-linux
  -androideabi/lib/ldscripts/armelf_linux_eabi.x -Wl,-dynamic-linker,
  /system/bin/linker -Wl,--gc-sections -Wl,-z,nocopyreloc -Wl,
  --no-undefined -Wl,-rpath-link=/home/zoe/android-ndk-r9d/platforms/
  android-8/arch-arm -L/home/zoe/android-ndk-r9d/platforms/platforms/
  android-8/arch-arm/usr/lib -L/home/zoe/AOSP_MAKO/out/target/product/
  mako/obj/lib/ -nostdlib /home/zoe/android-ndk-r9d/platforms/
  android-8/arch-arm/usr/lib/crtend_android.o /home/zoe/android-ndk
  -r9d/platforms/android-8/arch-arm/usr/lib/crtbegin_dynamic.o S-lc
  /home/zoe/android-ndk-r9d/toolchains/arm-linux-androideabi-4.6
  /prebuilt/linux-x86_64/lib/gcc/arm-linux-androideabi/4.6/
  libgcc.a /home/zoe/openvswitch-1.11.0/ofproto/libofproto.a
   /home/zoe/openvswitch-1.11.0/lib/libopenvswitch.a
   /home/zoe/openvswitch-1.11.0/lib/libsflow.a -lm
   -ldl -lgcc -lcrypto -lssl
```

2.4.3 Activate OpenVswitch in kernel 3.4

There is another way to have the module openvswitch.ko in the kernel for kernels of version 3.4 and above. The AOSP 4.2.2_1 branch is based on android linux kernel 3.4. Thus the OpenvSwitch is already in it. It just needs to be activated.

The OpenVswitch source code of android linux kernel exists in folder:

/home/zoe/AOSP_MAKO/msm/net/openvswitch

The Linux kernel comes with several configuration tools. Each one is run by typing make ¡something¿config in the top-level kernel source directory. (All make commands need to be run from the top-level kernel directory.

make config: This is the barebones configuration tool. It will ask each and every configuration question in order. The Linux kernel has a LOT of configuration questions. make menuconfig This command pops up a text-based menu-style configurator. change only the configuration options you care about.

Each of the configuration programs produces these end products: A file named .config in the top-level directory containing all your configuration choices.

- CONFIG_* = y it is compiled as build-in in kernel
- CONFIG_*=m it is compiled as a module in kernel

A file named autoconf.h in the include/linux/ directory defining (or not defining) the CONFIG_* symbols so that the C preprocessor knows whether or not they are turned on.

Activation of OpenVswitch in android linux kernel As for creating OpenVswitch module in kernel, we configured the hidden file .config manually. We set the CONFIG_OPENVSWITCH = m, as we want to build OpenVswitch as a module. Then we compile the kernel and the openvswitch.ko module is created in the folder:
/home/zoe/AOSP_MAKO/msm/net/openvswitch/

2.5 Build AOSP for NEXUS 5 - hammerhead and cross-compiling openvswitch

We should cross-compile the android tree and kernel from source and then cross-compile the openvswitch for android linux kernel.

We have used the following versions:

```
Android version:  android-4.4.2_r1(KitKat)
Android Kernel source git:  https://android.googlesource.com/kernel/msm.git
Kernel version:  3.4
Android Build: aosp_hammerhead-userdebug
Android NDK Version: arm-eabi-4.7
```

```
Openvswitch version: (Open vSwitch) 1.11.0
```

The directory structure of different source codes is the following(host OS: Ubuntu 13.04):

```
ANDROID_KERNEL: /home/zoe/AOSP_HAMMERHEAD/msm
ANDROID_SOURCE_CODE_PATH: /home/zoe/AOSP_HAMMERHEAD/
ANDROID_NDK_ROOT: home/zoe/AOSP_HAMMERHEAD/prebuilts/ndk/
OpenVswitch Source: /home/zoe/openvswitch-1.11.0/
ANDROID_SDK: /home/zoe/android-sdk-linux/
```

2.5.1 Building AOSP for NEXUS 5

Downloading the source

- Create an empty directory to hold your working files.Give it any name you like:

```
mkdir WORKING_DIRECTORY
cd WORKING_DIRECTORY
```

- Run repo init to bring down the latest version of Repo with all its most recent bug fixes. You must specify a URL for the manifest, which specifies where the various repositories included in the Android source will be placed within your working directory. To check out a branch specify it with -b:

```
repo init -u https://android.googlesource.com/platform/manifest -b android-4.4.2_r1
```

- When prompted, configure Repo with your real name and email address. To use the Gerrit code-review tool, you will need an email address that is connected with a registered Google account. Make sure this is a live address at which you can receive messages. A successful initialization will end with a message stating that Repo is initialized in your working directory. Your client directory should now contain a .repo directory where files such as the manifest will be kept.

To pull down the Android source tree to your working directory from the repositories as specified in the default manifest, run:

```
repo sync
```

Building the system

Choose a target

```
source build/envsetup.sh
lunch  aosp_hammerhead-userdebug
```

Building the code

```
make -j4
```

Building the kernel

Enter in the root directory

```
cd AOSP_HAMMERHEAD
```

Then download the appropriate kernel branch for the nexus 4:

```
git clone https://android.googlesource.com/kernel/msm.git
git branch -a
cd msm
```

And then we checkout the kernel source code:

```
git checkout android-msm-hammerhead-3.4-kitkat-mr1
```

We compiled the kernel with the android-ndk-r9d/toolchains/arm-linux-androideabi-4.6 toolchain. The compilation steps follow:

```
export PATH=/home/zoe/AOSP_KITKAT2/prebuilts/gcc/linux-x86/arm/arm-eabi-4.7/bin:$PATH

export ARCH=arm

export SUBARCH=arm

export CROSS_COMPILE=arm-linux-androideabi-

make hammerhead_defconfig

make
```

The kernel image file zImage will be created in the arch/boot/arm/ folder.

2.5.2 Cross compiling OpenVswitch

Cross-compiling OpenVswitch for the android linux kernel of the hammerhead failed with the same error messages of the maguro linux kernel. These errors appear since the linux kernel of the hammerhead may not be compatible with the OpenVSwitch. Changes are needed to become to the OpenVswitch source code. However, attempts for creating a patch changing the openvswitch code appropriately or finding a ready patch failed.

However, we can create the ovs module easily by activating it through the configuration parameters in the kernel. The activation procedure is the same as in the mako phone 2.4.3.

Despite we have the openvswitch.ko kernel module, it is necessary the openVSwitch source code to be cross-compiled successfully ,for the android linux kernel, in order to cross-compile the OVS user interface.

2.6 Building CyanogenMod for HTC EVO 4g (supersonic) and cross compile OpenVSwitch

2.6.1 Building CyanogenMod for HTC EVO 4g

Unlike the AOSP, the CyanogenMod offers the propriety binaries for the phone. Thus, we compile the CyanogenMod ROM from source code for the HTC EVO 4g. There are instructions about how to initiate your environment for building the Cyanogenmod source code[9]. The building instructions follows:

Create the directories

You will need to set up some directories in your build environment. To create them:

```
mkdir -p ~/bin
mkdir -p ~/android/system
```

Initialize the CyanogenMod source repository

Enter the following to initialize the repository:

```
cd ~/android/system/
repo init -u git://github.com/CyanogenMod/android.git -b ics
```

Download the source code

To start the download of all the source code to your computer:

```
repo sync
```

Get prebuilt apps

```
cd ~/android/system/vendor/cm
then enter:
./get-prebuilts
```

This should cause some prebuilt apps to be loaded and installed into the source code. Once completed, this does not need to be done again.

Prepare the device-specific code

After the source downloads, ensure you are in the root of the source code (cd /android/system), then type:

```
source build/envsetup.sh
breakfast supersonic
```

This will download the device specific configuration and kernel source for your device.

Extract proprietary blobs

Now ensure that your Evo 4G is connected to your computer via the USB cable and that you are in the /android/system/device/htc/supersonic directory (you can cd /android/system/device/htc/supersonic if necessary). Then run the extract-files.sh script:

```
./extract-files.sh
```

You should see the proprietary files (aka “blobs”) get pulled from the device and moved to the /android/system/vendor/htc directory.

Start the build

Time to start building! So now type:

```
croot
brunch supersonic
```

Download the kernel source code and build the whole system

```
mkdir -p ~/android/kernel
cd ~/android/kernel
git clone git://github.com/CyanogenMod/cm-kernel.git
cd cm-kernel
git branch -a
git checkout supersonic-kernel
```

```
export ARCH=arm
export SUBARCH=arm
export PATH=$PATH:/home/zoi/android/system/prebuilt/linux-x86/toolchain/arm
-eabi-4.4.3/bin
export CROSS_COMPILE=/home/zoi/android/system/prebuilt/linux
-x86/toolchain/arm-eabi-4.4.3/bin/arm-eabi-
export CCOMPILER=/home/zoi/android/system/prebuilt/linux-x86/toolchain/arm
-eabi-4.4.3/bin/arm-eabi-
```

Configure and compile the kernel source code. All the configuration files exist in arch/arm/configs folder, similarly the supersonic_defconfig)

```
make ARCH=arm supersonic_defconfig

make ARCH=arm CROSS_COMPILE=/home/zoi/android/system/prebuilt/linux
-x86/toolchain/arm-eabi-4.4.3/bin/arm-eabi- -j‘grep ’processor’ /proc/cpuinfo | wc -l‘
```

Then, we copy the kernel file to the folder where the cyanogenmod searches for the kernel image.

```
cp /home/zoi/android/kernel/cm-kernel/arch/arm/boot/zImage
 /home/zoi/android/system/device/htc/supersonic/kernel
```

Building the system

```
cd android/system/
. build/envsetup.sh && brunch supersonic
```

During the building the following error occurs:

```
Error
collect2: ld returned 1 exit status
make: *** [out/target/product/supersonic/obj/SHARED_LIBRARIES/libcameraservice_intermediates
/LINKED/libcameraservice.so] Error 1
```

The problem is solved by pulling the libcamero.so file:

```
adb pull  system/lib/libcamera.so
```

and place it to the folder vendor/htc/supersonic/proprietary of the Cyanogenmod tree.

2.6.2 Cross compiling OpenVswitch

The version of openvswitch is 2.0.0 and the android linux kernel is located in the folder android/system. The directory structure of different source codes is the following(host OS: Ubuntu 13.04):

```
ANDROID_KERNEL=/home/zoi/android/kernel/cm-kernel/
ANDROID_SOURCE_CODE_PATH = /home/zoi/android/system/
ANDROID_NDK_ROOT=/home/zoi/android/system/prebuilt/linux-x86/toolchain/arm-eabi-4.4.3/
OpenVSwitch Source: /home/zoi/openvswitch-2.0.0
ANDROID_SDK: /home/zoe/android-sdk-linux/
```

Configure and compile the openvswitch:

```
export PATH=$PATH:"/home/zoi/android/system/prebuilt/linux-x86/toolchain/arm-eabi-4.4.3/bin"

./configure ovs_cv_use_linker_sections=no -with
-l26=/home/zoi/android/kernel/cm-kernel KARCH=arm --with-rundir=/data/local/var


ARCH=arm CROSS_COMPILE=arm-eabi- make
```

The OpenVSwitch was compiled successfully, and the openvswitch.ko kernel module was created in the datapath/linux folder. However, we had no success to compile the ovs user interface.

2.7 Build AOSP for x86 platform and cross-compiling openvswitch

2.7.1 Building AOSP for -x86 platform

Firstly, we have followed the instruction for initializing the environment at the section Initializing environment 2.2.1. Then, we downloaded the kitkat branch:

```
mkdir android-x86
cd android-x86
repo init -u http://git.android-x86.org/manifest -b kitkat-x86
repo sync
```

The folder with the kernel source already exists in android open source x86 [10]. The Android build system doesn't compile kernel on-fly. It just contains a prebuilt kernel binary which will be added to the target image. This approach may be good enough for the arm emulator target, but not suitable for x86 platforms. The x86 platforms have various hardware. The kernel binary and its modules may need to be adjusted at compile time or runtime. The android-x86 gives the ability to build kernel and modules by a predefined or customized config during the building process.

We made our own my_defconfig file configuring the kernel and placed it in the folder kernel/arch/x86/configs/. We set in the my_def_config file the configuration parameter CONFIG_OPENVSWITCH = m so as the openvswitch to be built as module. Then, we build whole the sustem along with the kernel specifying the kernel configuration file:

```
cd android-x86
make clean
make -j4 iso_img TARGET_KERNEL_CONFIG:=my_defconfig
```

The build process was completed successfully. The openvswitch.ko was created in the folder /home/zoi/android-x86/android-x86_kitkat/out/target/product/x86/obj/kernel/net/openvswitch. The image that we will flash on the target is in iso format. After the build process, the android-x86.iso is created in the folder /home/zoi/android-x86/android-x86_kitkat/out/target/product/x86/obj/kernel/net/openvswitch. Then, we flashed the android-x86.iso on a virtual machine and noticed that the kernel module was placed in this folder /system/lib/modules/3.10.30-andeois-x86+/kernel/net/openvswitch/.

2.7.2 Cross-compiling OpenVswitch

Despite, we activated the openvswitch module successfully, we did not achieve to cross-compile openvswitch user interface for android.

2.8 Conclusion

To sum up, we have cross-compiled OpenVswitch for four mobile devices and one virtual machine based on x86 platform. We have failed to get the openvswitch.ko kernel module for the HTC Evo 4G and the Galaxy Nexus. However, we have cross compiled the OVS kernel module successfully for the Nexus 4, Nexus 5 and for x86 platforms. Also, we have cross-compiled some OVS user interface utilities successfully for the Nexus 4. The following user interface utilities have been cross-compiled successfully:

- ovs-dpctl
- ovs-ofctl
- ovs-vswitchd
- ovsctl

The following user interface utilities have not been cross-compiled successfully:

- ovsdb-server
- ovsdb-tool

	HTC Evo 4G Gingerbread	Galaxy Nexus Jelly bean	Nexus 4 Kit Kat	Nexus 5 Kit Kat	Android-x86 Kit Kat
OVS kernel module	No	No	Yes	Yes	Yes
OVS User interface	No	No	Partially Yes	No	No

FIGURE 2.2: This figure shows if the openvswitch kernel module and its user interface compiled successfully for each device

Chapter 3

Technologies Overview

In this section we will see the technologies used to develop the NitosTools application with the intention of the reader understands better the issues discussed.

Android applications are written in Java Programming language . Android is designed with the maximum amount of modularity in mind. This modularity makes it easy for developers to exchange functionality between their applications, which is a central concept of the open source paradigm upon which Android is based.

There are four main types of components that can be (but do not need to be) used within an Android application:

- Activities handle the UI to the smartphone screen.
- Services handle background processing.
- Broadcast receivers handle communication in your apps.
- Content providers handle data and database management issues.

3.1 Activities

An Activity is an application component that provides a screen with which users can interact in order to do something, such as dial the phone, take a photo, send an email, or view a map. Each activity is given a window in which to draw its user interface. If an application has more than one activity, then one of them should be marked as the activity that is presented when the application is launched.

3.1.1 Activity's Lifecycle

An application can consists of multiple activities [1]. One activity may trigger another activity.Each time a new activity starts, the previous activity is stopped, but the system preserves the activity in a stack (the "back stack"). When a new activity starts, it is pushed onto the back stack and takes user focus. The back stack abides to the basic "last in, first out" stack mechanism, so, when the user is done with the current activity and presses the Back button, it is popped from the stack (and destroyed) and the previous activity resumes.

When an activity is stopped because a new activity starts, it is notified of this change in state through the activity's lifecycle callback methods. There are several callback methods that an activity might receive, due to a change in its state—whether the system is creating it, stopping it, resuming it, or destroying it—and each callback provides you the opportunity to perform specific work that's appropriate to that state change. For instance, when the activity is paused, it is a good practice to save the variables you want so as to restore them later. The onStop() method is not guaranteed to be called, thus onPause is the appropriate method for storing state.

To create an activity, you must create a subclass of Activity (or an existing subclass of it). In your subclass, you need to implement callback methods that the system calls when the activity transitions between various states of its lifecycle, such as when the activity is being created, stopped, resumed, or destroyed. All activities will implement onCreate(Bundle) to do their initial setup; many will also implement onPause() to commit changes to data and otherwise prepare to stop interacting with the user. Thus, the two latter callback methods are the most important:

- onCreate()
  - Called when the activity is first created. This is where you should do all of your normal static set up: create views, bind data to lists, etc. This method also provides you with a Bundle containing the activity's previously frozen state, if there was one.
- onPause()
  - Called when the system is about to start resuming a previous activity. This is typically used to commit unsaved changes to persistent data, stop animations and other things that may be consuming CPU, etc. Implementations of this method must be very quick because the next activity will not be resumed until this method returns.

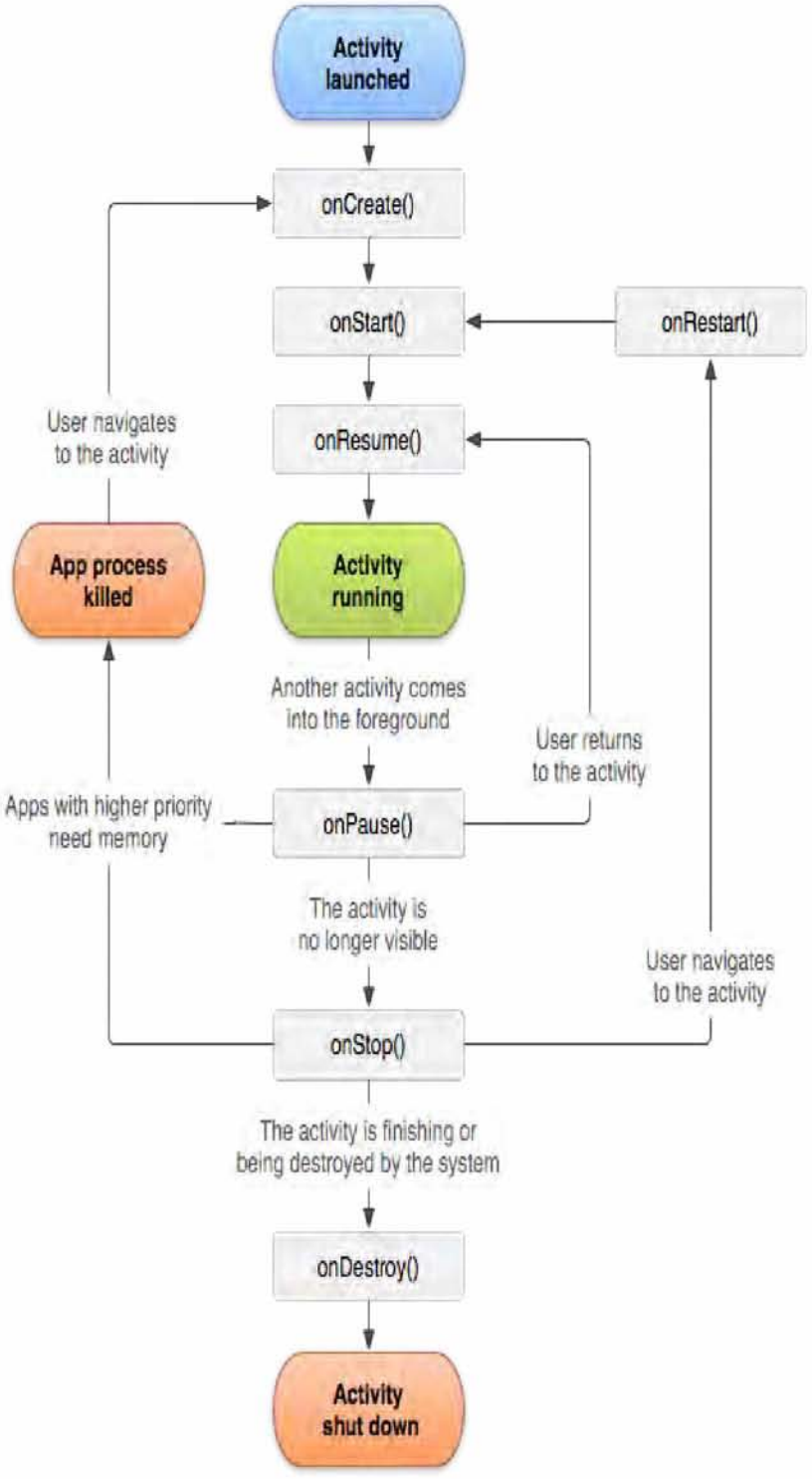

FIGURE 3.1: The diagram shows the important state paths of an Activity[1].

There are three key loops in monitoring within your activity:

- The **entire lifetime** of an activity happens between the first call to onCreate(Bundle) through to a single final call to onDestroy(). An activity will do all setup of "global" state in onCreate(), and release all remaining resources in onDestroy().
- The **visible lifetime** of an activity happens between a call to onStart() until a corresponding call to onStop(). During this time the user can see the activity on-screen, though it may not be in the foreground and interacting with the user. Between these two methods you can maintain resources that are needed to show the activity to the user.
- The **foreground lifetime** of an activity happens between a call to onResume() until a corresponding call to onPause(). During this time the activity is in front of all other activities and interacting with the user.

3.2 Fragments

Fragments represent a re-usable portion of an Activity's user interface. You can combine multiple fragments in a single activity to build a multi-pane layout user interface and reuse fragments in multiple activities. In other words, a fragment is a modular section of an activity. A fragment is a sort of like a "sub activity" that you can reuse in different activities. It has its own lifecycle, receives its own input events and can be added or removed while the activity is running.

A fragment must always be embedded in an activity and the fragment's lifecycle is directly affected by the host's activity's lifecycle. For instance, when the activity is paused, all fragments in the activity will also be paused. However, while the activity is running(It is in the resumed lifecycle state), you can manipulate each fragment independently, such as add or remove them. Fragment transactions are used to remove and replace fragments associated with the action bar tabs, such as:

- ft.remove(fragment); remove a fragment from the UI
- ft.replace(R.id.fragment_container, fragment); replace one fragment (or view) with another.

These fragment transactions allow the activity to manage the back stack - a back stack entry in the activity is a record of the fragment transaction that occurred. This,

for example, allows the user to reverse a fragment transaction (navigate backwards), by pressing the back button.

When you add a fragment as a part of your activity layout, it lives in a ViewGroup inside the activity's view hierarchy and the fragment defines its own view layout. There are two ways to add a fragment into your activity layout. You can added statically by declaring the fragment in the activity's layout file, as a ¡fragment¿ element or dynamically from your application code by adding to an existing ViewGroup.

To create a fragment, you must create a subclass of Fragment. The Fragment class has code that looks a lot like an Activity. It contains callback methods similar to an activity, such as onCreate(), onStart(), onPause(), and onStop().

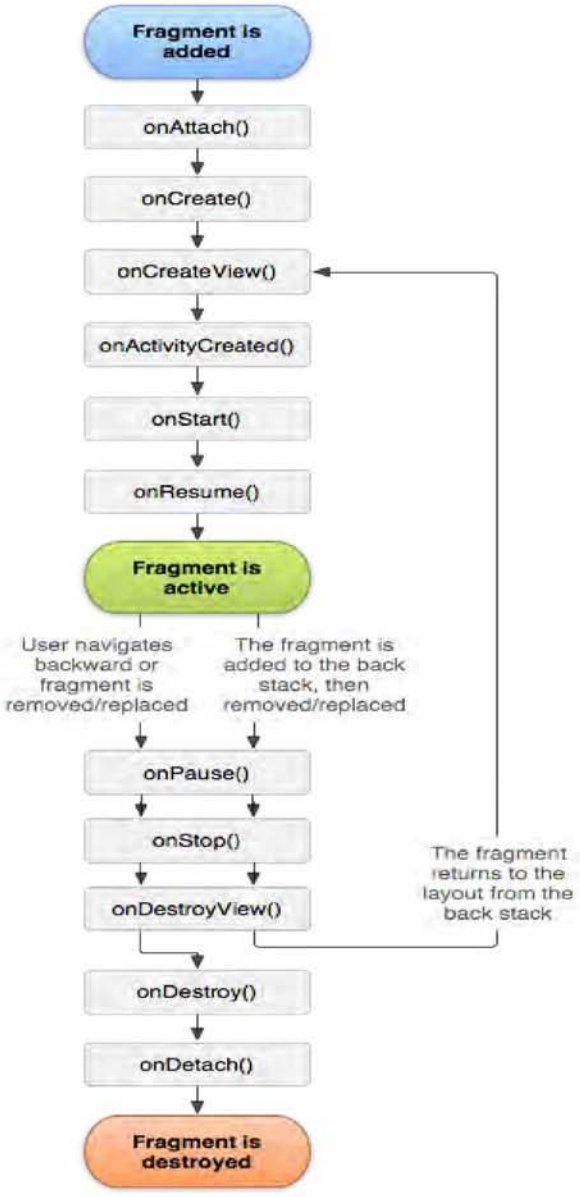

FIGURE 3.2: The lifecycle of a fragment[1] (while its activity is running).

Most applications should implement at least the following three methods for every fragment, but there are several other callback methods you should also use to handle various stages of the fragment lifecycle. Usually, you should implement at least the following lifecycle methods:

- **onCreate()**
  - The system calls this when creating the fragment. Within your implementation, you should initialize essential components of the fragment that you want to retain when the fragment is paused or stopped, then resumed.
- **onCreateView()**
  - The system calls this when it's time for the fragment to draw its user interface for the first time. To draw a UI for your fragment, you must return a View from this method that is the root of your fragment's layout. You can return null if the fragment does not provide a UI.
- **onPause()**
  - The system calls this method as the first indication that the user is leaving the fragment (though it does not always mean the fragment is being destroyed). This is usually where you should commit any changes that should be persisted beyond the current user session (because the user might not come back).

3.3 User Interface

Up to this point, we have seen that the basic unit of an Android application is an Activity. The Activity displays the user interface of your application, which may contain widgets like buttons, labels, text boxes e.t.c A layout defines the visual structure for a user interface, such as the UI for an activity . You can declare a layout in two ways:

- Declare UI elements in XML. Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts.
- Instantiate layout elements at runtime. Your application can create View and ViewGroup objects (and manipulate their properties) programmatically.

Some of the most common layouts are:

- **LinearLayout** is a layout that aligns all children in a single direction, vertically or horizontally.
- **RelativeLayout** is a layout that every element arranges itslef relative to other elements or a parent element.
- **TableLayout** is a layout that is divided into rows and columns.
- **FrameLayout** is designed to block out an area on the screen to display a single item.
- **ScrollView** is a layout that can be scrolled by the user, allowing to be larger than the physical display.

3.4 Action Bar

The action bar is a window feature that identifies the user location, and provides user actions and navigation modes. Using the action bar offers your users a familiar interface across applications that the system gracefully adapts for different screen configurations.

FIGURE 3.3: action bar[1]

The action bar provides several key functions:

- Provides a dedicated space for giving your app an identity and indicating the user's location in the app.
- Makes important actions prominent and accessible in a predictable way (such as Search).

- Supports consistent navigation and view switching within apps (with tabs or drop-down lists).

3.5 Different ways to share data between Activities/Fragments

There is no perfect storage option [2] since it depends on the data that we want to share. There are two ways of sharing data: passing data in the intent's extras or saving it somewhere else. If data is primitives, Strings or user-defined objects: send it as part of the intent extras (user-defined objects must implement Parcelable). If passing complex objects save an instance in a singleton somewhere else and access them from the launched activity.

- **Primitive Data Types** To share primitive data between Activities/Services in an application, use Intent.putExtras(). For passing primitive data that needs to persist use the Preferences storage mechanism.
- **Non-Persistent Objects** For sharing complex non-persistent user-defined objects for short duration, the following approaches are recommended:
  - **Singleton class** You can take advantage of the fact that your application components run in the same process through the use of a singleton. This is a class that is designed to have only one instance. It has a static method with a name such as getInstance() that returns the instance; the first time this method is called, it creates the global instance. Because all callers get the same instance, they can use this as a point of interaction.
  - **Application singleton** Each Android application can have at most one android.app.Applicationassociated with it. You are responsible for sub-classing the Application Class, and it is used to maintain a global state of the application across all Activites. Conceptually, you can think of it as a non-static singleton its life cycle being managed by Android OS.
  - **A HashMap of WeakReferences to Objects** You can also use a HashMap of WeakReferences to Objects with Long keys. When an activity wants to pass an object to another activity, it simply puts the object in the map and sends the key (which is a unique Long based on a counter or time stamp) to the recipient activity via intent extras. The recipient activity retrieves the object using this key.

  - **A public static field/method** An alternate way to make data accessible across Activities/Services is to use public static fields and/or methods. You can access these static fields from any other class in your application. To share an object, the activity which creates your object sets a static field to point to this object and any other activity that wants to use this object just accesses this static field.

- **Persistent Objects** Even while an application appears to continue running, the system may choose to kill its process and restart it later. If you have data that you need to persist from one activity invocation to the next, you need to represent that data as state that gets saved by an activity when it is informed that it might go away. For sharing complex persistent user-defined objects, the following approaches are recommended:

  - Application Preferences
  - Files
  - contentProviders
  - SQLite DB

Chapter 4

System Design

This chapter presents the design of the mobile application.

4.1 System Architecture

The system architecture has been illustrated on Figure 4.1. From the left-hand side an Android device (not necessarily a smartphone, it could be a tablet) with the NitosTools application already installed communicates with the web service using a special RESTful API. The application sends HTTP requests with GET/PUT/POST/DELETE method headers and receives well formatted JSON responses. REST objects can be exchanged both ways.

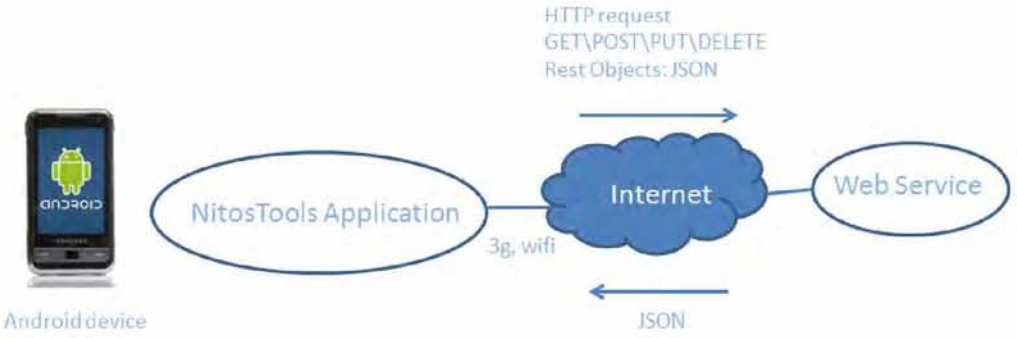

FIGURE 4.1: System architecture

4.2 Application Architecture

In this section we will describe the NitosTools application's architecture. Firstly, we will present a simplified diagram presenting the main classes of the system in Figure 4.2. It is a reference to the extended class diagram which includes the attributes and operations of the classes. The extended class diagram is available in Appendix A.

Each class which has the suffix Activity represents a screen of the mobile application. For example, LoginScreenActivity.class is the login screen. The classes that has suffix Fragment, such as SchedulerChooserFragment, are fragments.

The diagram shows the relationships (dependencies) between the classes. When an activity class depends on another activity class in our design, it means that the independent class is called by the depended class.

The class diagram does not include the inner classes in the MyReservationsActivity, SchedulerChooserFragment and ReserveResourcesFragment that parse the JSON files, so as the class diagram to be easily readable. An extended diagram, with the classes that sends requests to the servers and parses the JSON files, is available in the Appendix A.

NitosTools consists of four activities:

- LoginScreenActivity: validates the user.
- MainMenuActivity: displays a list of all the testbed tools.
- NitosSchedulerActivity: creates a tabbed user interface.
- MyReservationsActivity: displays all the user's reservations and offers the ability for canceling them.

Apart from the activities, the application consists of eight fragments:

- SchedulerChooserFragment displays the user's interface for choosing the reservation's parameters(slice, date, time, duration) and checking the available resources.
- DatePickerFragment: is fired through the SchedulerChooserFragment for the date selection.
- TimePickerFragment: is fired through the SchedulerChooserFragment for the time selection.

- OutdoorTestbedFragmen: displays the outdoor testbed
- IndoorTestbedFragment: displays the indoor testbed
- AvailableResourcesFragment: displays a UI displaying all available nodes. The user can reserve the resources through this fragment.
- CheckResourcesFragment: a nested fragment to the AvailableNodesFragment, which displays all the nodes of a specific type in a listview of checkboxes.
- ReserveResourcesFragment: performs the reservation process and in case of a successful reservation displays an appropriate message.

There are also two classes performing the requests to the server:

- TestbedHttpClient: This class contains methods for connecting with the server and making GET and POST requests.
- GlobalData: This class contains all the global data for the application
- Reservation: This class contains information(date and time) for a reservation.
- Constants: Constants class is responsible for all the constants of the application.

4.3 Description of classes and fragments

4.3.1 Main Activities

4.3.1.1 LoginScreenActivity extends Activity

LoginScreenActivity is the main Activity as we have defined it on the Manifest file, which means that it is the first activity that is created when NitosTools is launched. This activity accepts user name and password from the user , and then checks their validity. If the username and password are valid, the LoginScreenActivity redirects the user to the MainMenuActivity. There is a "Keep me signed in" checkbox in the LoginScreenActivity's user interface. If the user does not want to login every time launches the application, he should check this checkbox. When the "Keep me signed in" is checked, the user is redirected to the MainMenuActivity's screen.

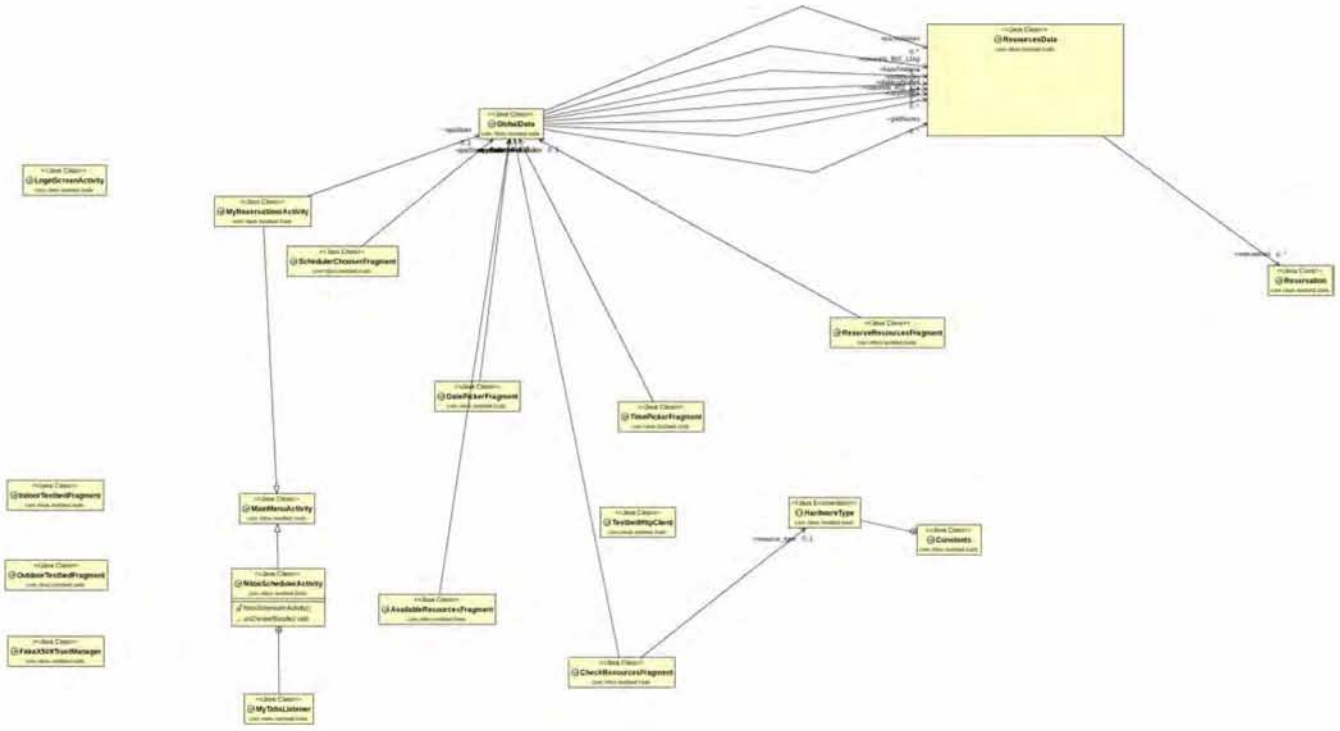

FIGURE 4.2: NitosTools Application

4.3.1.2 MainMenuActivity extends ActionBarActivity

MainMenuActivity displays a list with all Testbed Tools. Only the Nitos scheduler tool is implemented. When the Nitos scheduler list item is clicked, the user is redirected to the NitosSchedulerActivity. The action bar of the application is visible in the MainMenuActivity activity. Two important actions(User's reservation history and Log out) are made prominent and accessible through the action bar.

4.3.1.3 NitosSchedulerActivity extends MainMenuActivity

Three fragment tabs are added to the NitosSchedulerActivity's Action Bar. The tabs,on tab click, will show the selected fragments. These tabs have as text prompts "Select", "Outdoor",and "Indoor", and the fragments hosted :

- "Select" tab
  - The tab with the prompt text "Select" hosts the SchedulerChooserFragment which is responsible for the selection of the slice, the date, the time and the duration of the reservation. These tab hosts, also, nested fragments which will be described later.
- "Outdoor" tab

  - This tab hosts the OutdoorTestbedFragment. This fragment is responsible for displaying the outdoor testbed with all the nodes. The kind of each node is determined from its color.

- "Indoor" tab

  - This tab hosts the IndoorTestbedFragment. This fragment is responsible for displaying the indoor testbed with all the nodes. All nodes , in the indoor testbed, are Icarus nodes.

4.3.1.4 MyReservationsActivity extends MainMenuActivity

MyReservationsActivity is fired when the user icon on action bar is pressed. This Activity is responsible for displaying all the resources being reserved from the user. To get the leases, MyReservationsActivity makes an HTTP Get Request to the server returning the leases in JSON format. Then, the activity parses the JSON file and retrieves all the leases with account names equals user's account names. While the activity parses the JSON file, it creates the UI dynamically. For each resource, it prints the date and time in which the resource is reserved, the name of the resource, and a "cancel" button to cancel the reservation.

4.3.2 Other Classes

We will describe some other classes not being activities, but have a supportive purpose to the activities and fragments of the application:

4.3.2.1 TestbedHttpClient

The TestbedHttpClient class provides HTTP services to the application. In particular, it has two methods, one for making HTTP GET request, and one for making HTTP POST Request to the server.

4.3.2.2 GlobalData extends Application

The GlobalData class is a subclass of the android.app.Application class, thus retains a global state of the NitosTools application. It contains all the resources which should be global across all activities and fragments. In particular, the resources in the testbed, and all the resources which are available ,during the date and time entered by the user,

are declared in this class. Also, the GlobalData class provides accessors and mutators methods for the resources.

4.3.2.3 Constants

This class is used to keep constant values.

- String **BASE_URL**: this is a string value referring to the base URL used for making HTTP requests. This constant is only used by the TestbedHttpClient class.
- String **CHECKBOXES_PREFERENCES** is a string value used by the Shared-Preferences in the AvalaibleResourcesFragment and CheckResourcesFragment.
- String **LOGIN_PREFS**: this is a string value used by the SharedPreferences in the LoginScreenActivity and MainMenuActivity.
- String **LOWER_CHANNEL_802_11a, UPPER_CHANNEL_802_11a , LOWER_CHANNEL_802_11bg , UPPER_CHANNEL_802_11bg**: These String constants are used for categorizing channels in protocols taking into account its number. These constants are used in the SchedulerChooserFragment.
- enum **HardwareType**: The HardwareType is an enumeration type referring to the set of the resources.The set of the resources is the following : ORBIT, GRID, USRP, DISKLESS, ICARUS,BASE_STATIONS,CHANNELS_802_11A, CHANNELS_802_11BG.
- String **TAG_RESOURCE_RESPONSE, TAG_RESOURCES, TAG_COMPONENTS, TAG_COMPONENT**: these are string constants which are used for referring to some tags of the JSON files through the parsing.

4.3.3 Fragments

4.3.3.1 SchedulerChooserFragment extends Fragment

The SchedulerChoooserFragment is hosted in the NitosSchedulerActivity's first tab. This fragment displays the UI for setting the required parameters for the reservation. Firstly, it displays the local date and time. It contains two buttons , "Pick a date" and "Pick time", for picking the date and time for the reservation. When the user presses

the button "Pick a date" and "Pick time", the DatePickerFragment and the TimePickerFragment are inflated respectively. Also, it contains a spinner to select the duration of the reservation.

When the user sets all the parameters and is ready to see the available nodes, he should press the "Check Available Resources" Button. When the latter button is pressed, three HTTP GET requests are performed for receiving the JSON files for the nodes, the channels and the leases. Firstly, it parses the JSON file for the nodes, and sets each node to a global data structure, defined in the GlobalData class, accordingly to the node's Hardware Type. Also, it parses the JSON file for the channels and sets each channel to a global data structure accordingly to its protocol. Finally, the SchedulerChooserFragment parses the JSON file for the leases and stores each lease in the related resource's data structure. All the global data structures, defined in the GlobalData class, will be described in detail in Chapter 5. After the clicking of the "Check Available Resources" button, all the previous processing is performed and then the AvailableResourcesFragment is inflated. The UI of this fragment is presented in the Figure 4.3 figure.

4.3.3.2 DatePickerFragment extends DialogFragment implements DatePickerDialog.OnDateSetListener

DatePickerFragment is launched by the SchedulerChooserFragment through the "Pick a date" button. Each time the button is pressed, a dialog appears. The first time the "Pick a date" is pressed, the dialog displays the current local date depending on where the mobile phone is located.

4.3.3.3 TimePickerFragment extends DialogFragment implements TimePickerDialog.OnTimeSetListener

TimePickerFragment is launched by the SchedulerChooserFragment through the "Pick time" button. Each time the button is pressed, a dialog appears with 30 minute intervals. The first time the "Pick time" is pressed, the dialog displays "00:00" time.

4.3.3.4 OutdoorTestbedFragment extends Fragment

The OutdoorTestbedFragment is hosted in the NitosSchedulerActivity's second tab. It just displays the outdoor testbed categorizing the resources with colors according to their hardware type. This fragment's interface is presented on the Figure 4.4

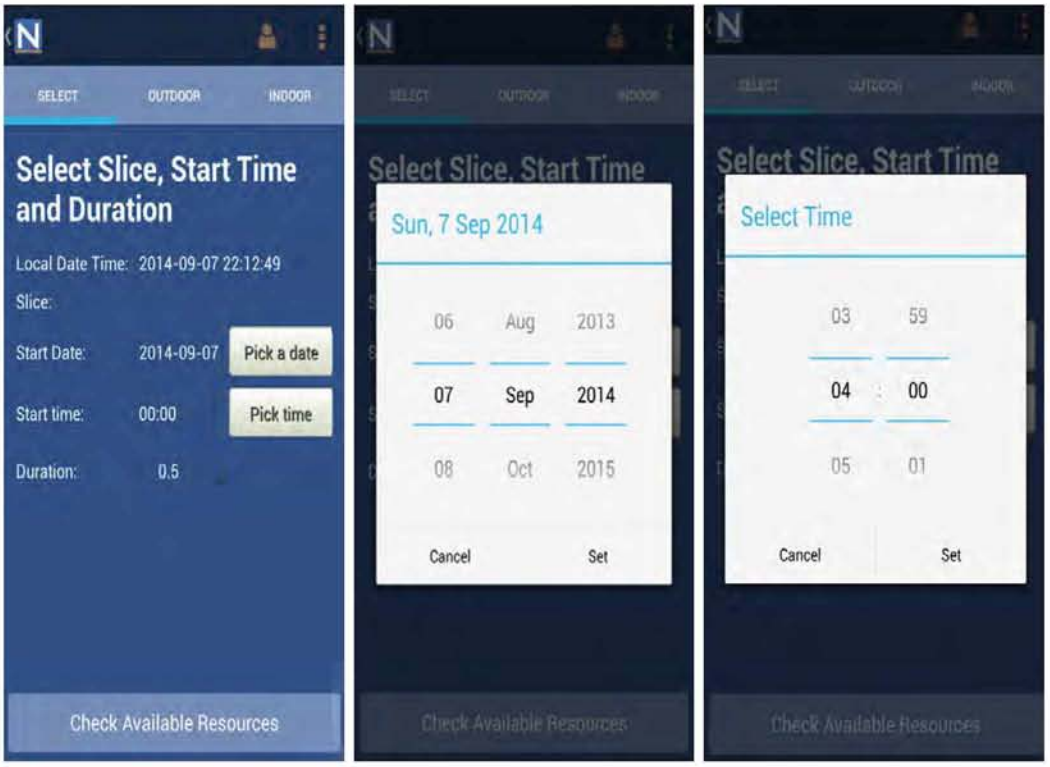

FIGURE 4.3: The SchedulerChooserFragment's UI is presented on this figure at the left corner. The DatePickerFragment and the TimePickerFragment dialogs , inflated from SchedulerChooserFragment's buttons, are presented next to the latter respectively.

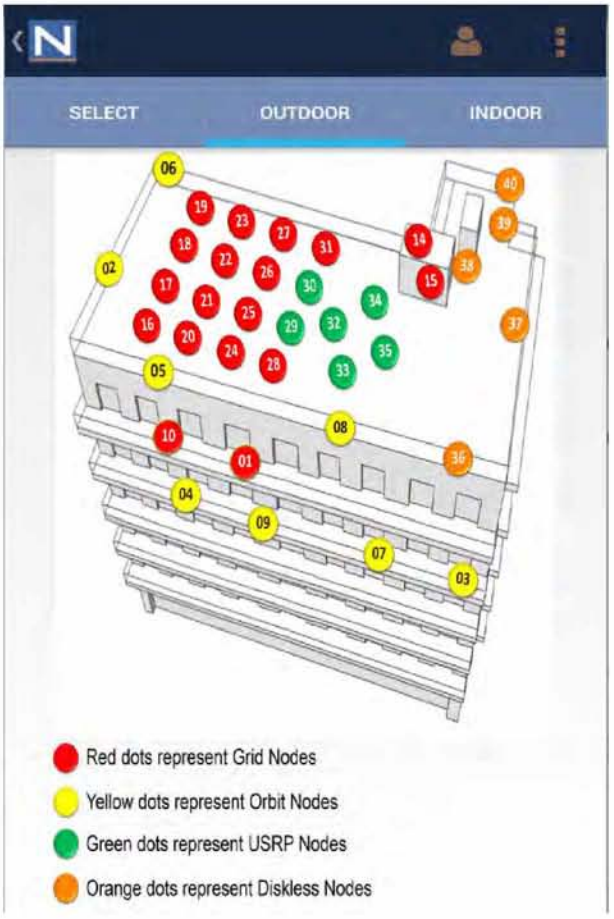

FIGURE 4.4: The OutdoorTestbedFragment's UI is presented on this figure.

4.3.3.5 IndoorTestbedFragment extends Fragment

The IndoorTestbedFragment is hosted in the NitosSchedulerActivity's third tab. It just displays the indoor testbed. Only icarus nodes are located in the indoor testbed.

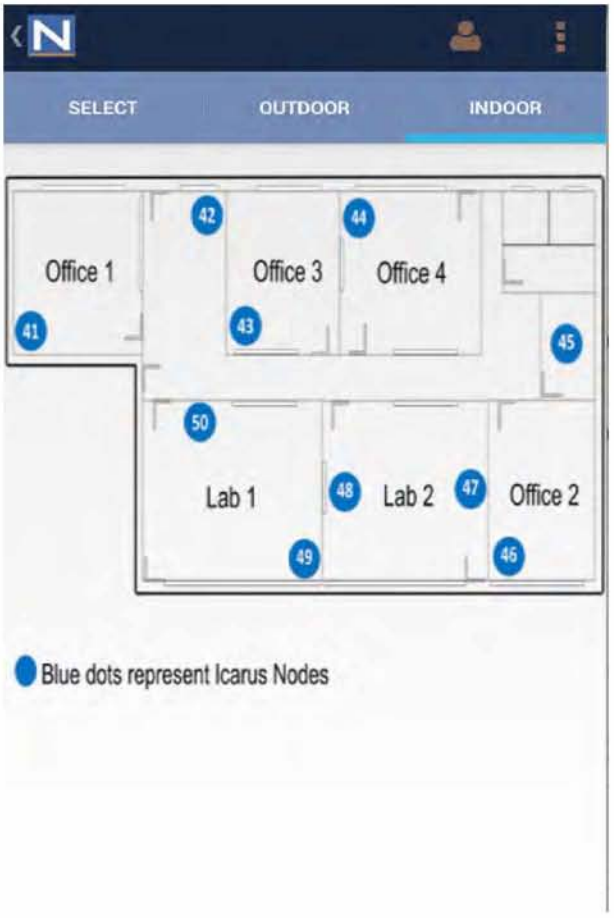

FIGURE 4.5: The IndoorTestbedFragment's UI is presented on this figure.

4.3.3.6 AvailableResourcesFragment extends Fragment

The AvailableResourcesFragment fragment is inflated from the "Check Available Resources" button of the SchedulerChooserFragment fragment. It displays all the available resources. Its UI consists of a list displaying all the categories of the resources, the nested CheckResourcesFragment fragment displaying the resources of the matching category in a listview of checkboxes, and a "Reserve" button inflating the ReserveResourcesFragment fragment performing the reservation process. The CheckResourcesFragment retains the checkboxes state. Thus, while the user navigates through the list and checks the checkboxes of each category, the checkboxes state' being lost. This fragment's interface is presented on the Figure 4.6.

4.3.3.7 CheckResourcesFragment extends Fragment

The CheckResourcesFragment is hosted in the AvailableResourcesFragment. It is placed on the right of the listView, inflated each time a listview item of the AvailableResources-Fragment is pressed, and the corresponding resources are displayed as checkboxes in the CheckResourcesFragment. This fragment keeps the checkboxes' state, so as the user will check all the resources he wants, and then click the "Reserve" button to inflate the ReserveResourcesFragment performing the reservation process.

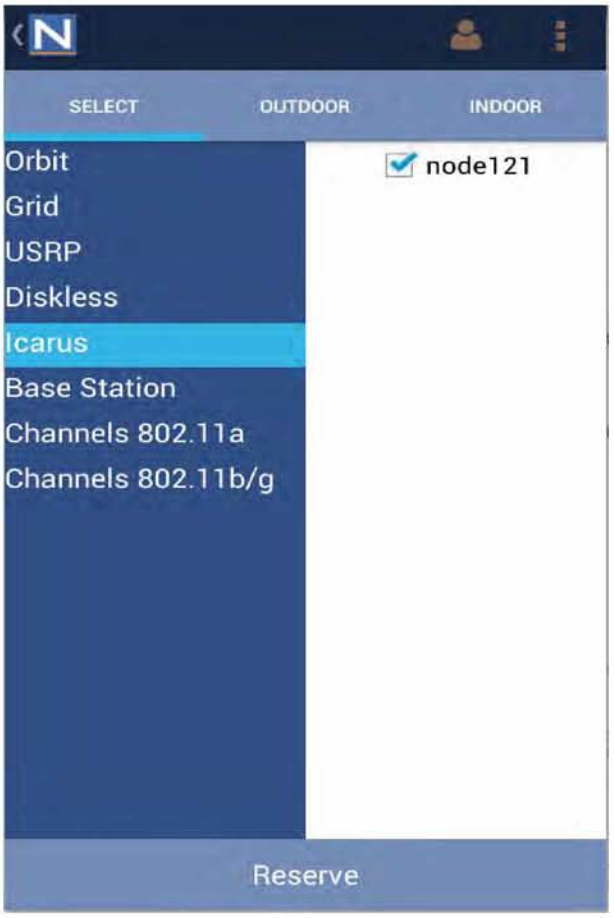

FIGURE 4.6: The AvailableResourcesFragment's UI is displayed on this figure. The CheckResourcesFragment fragment which is a nested fragment to the previous one is the white one with the checkboxes.

4.3.3.8 ReserveResourcesFragment extends Fragment

The ReserveResourcesFragment is inflated from the "Reserve" button's pressing of the AvailableResourcesFragment. It checks the state of all the checkboxes, and then makes a JSON message with all the selected resources, the dates and times when the reservation starts and ends, and other essential elements. Finally, it sends an HTTP POST request to the server with the JSON message in order to perform the reservation. This fragment's interface is presented on the Figure 4.7.

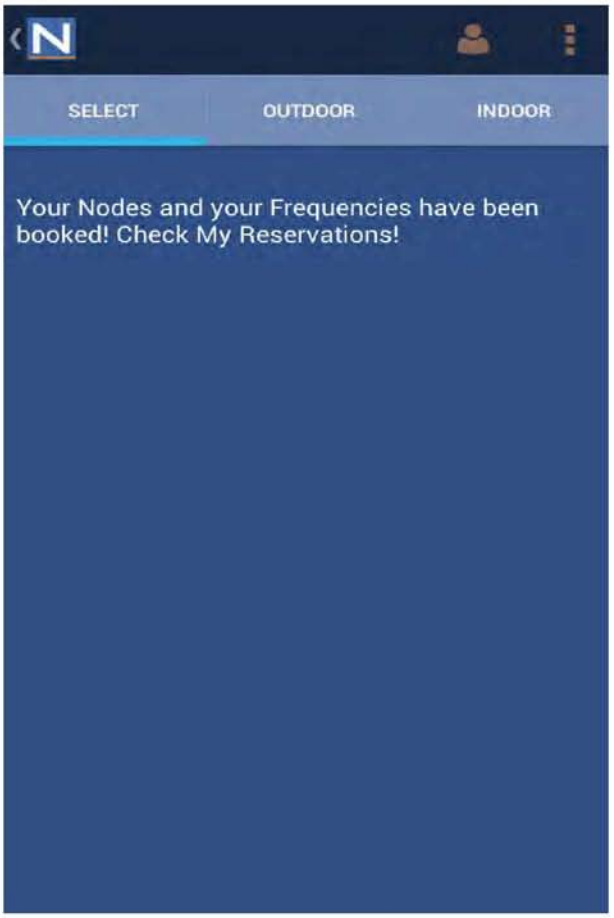

FIGURE 4.7: The ReserveResourcesFragment's UI is presented on this figure.

Chapter 5

Implementation

In this chapter, we will discuss some implementation details of some classes we described in the previous chapter.

5.1 GlobalData class

The implementation of GlobalData class is needed to keep data global across the application. This class is an application singleton. Each Android application can have at most one android.app.Application associated with it. Thus, the GlobalData class subclasses the Application class. There are also other ways to share data between the fragments and activities, such as exchanging data through interfaces. However, this kind of sharing data is suitable for primitive types. Our data structures are complex belonging to Java Collections, thus we use the application singleton for sharing the data. The global data structures are referring to:

- all resources existing at testbed and their data type is HashMap<String, ResourcesData>
- all resources being available at the given user's date ad time and their data type is TreeMap<String, String>
- checkboxes associated to the available resources at the given time and their data type is ArrayList<CheckBoxes>
- the date and time entered by the user

The GlobalData class also provides accessors and mutators methods for each data structure. The GlobalData class diagram is shown at Figure 5.1.

FIGURE 5.1: GlobalData Class Diagram

5.2 Fragments using the global data

5.2.1 SchedulerChooserFragment

When the user selects date, time and duration for the reservation, and then clicks the "Check Available Resources" button, the SchedulerChooserFragment fragment makes request to the server and then stores all the existing resources to the data structures defined in GlobalData class. There are eight different kinds of resources(orbit, grid, usrp, diskless, and icarus nodes base Stations, channels for 802_11a and 802_11bg protocol). So, there are eight data structures of type HashMap<String, ResourcesData>. The SchedulerChooserFragement makes three HTTP GET requests to get the nodes, the channels, and the leases in JSON format respectively. Then, while it parses the JSON files, it stores the resources to the HashMap<String, ResourcesData> data structures.

The HashMap<String, ResourcesData> data structure has as a key a String representing the node name and as a value an object of the ResourcesData class. The ResourcesData class has as instance fields a String uuid needed in case of an HTTP POST request , and an ArrayList of Reservation objects. The Reservation class contains instance fields for date and time about when a reservation starts and finishes. Each resource may be reserved many times, thus has to be associated with an ArrayList of Reservation objects. In this way, we can compare the user's given date and time with the ArrayList<Reservation> to check each resource's availability. The figure representing the HashMap<String, ResourcesData> structure is Figure 5.2.

In particular, the SchedulerChooserFragment makes the first HTTP request to get the JSON String with all the nodes and then parses the String while storing the node name for each node to the key of the HashMap<String, ResourcesData>, selecting the data structure according to node's hardware type. Secondly, this fragment makes the second HTTP request to get the JSON String with all the Channels, and stores them similarly to the nodes' procedure. Finally, the SchedulerChooserFragment fragment makes an HTTP GET request to get the JSON string about all the leases, namely all the reservations. Then, it parses the leases String, and stores, for each resource, the starting dates and times, and the ending dates and times to a Reservation object and adds this object to the ArrayList of the ResourcesData object of the matching HashMap<String, ResourcesData>.

5.2.2 AvailableResourcesFragment

After the "Check Available Resources" is pressed, the AvailableResourcesFragment fragment is inflated. As mentioned in 3, the AvailableResourcesFragment UI has a listview with all the categories of the resources. Each time the user clicks a listview item, the AvailableResourcesFragment computes which resources of the matching category are available at the given date and time and stores them in a TreeMap<String, String>(see Figure 5.3). The key of the TreeMap represents the resource name and the value represents the UUID. The UUID will be useful in the case of the reservation of the resource. The AvailableResourcesFragment has a nested fragment CheckResourcesFragment displaying the corresponding available resources with checkboxes each time a listview item is clicked. The reason we choose the TreeMap data structure is that it sorts the items by key with the red-black tree algorithm. In this way, the CheckResourcesFragment fragment displays the checkboxes sorted while parsing the available resources in a TreeMap. Also, while the user navigate across the listview's items and checks the checkboxes, the checkboxes' state is retained through the shared preferences 2.

5.2.3 ReserveResourcesFragment

When the user has checked all the desired resources, he presses the "Reserve" button of the "AvailableResourcesFragment" and the "ReserveResourcesFragment" is inflated. This fragment checks the global checkboxes structures in order to examine which resources are checked in the checkboxes. Then, for each resource checked, retrieves the UUID from the TreeMap, and formats a JSON message to perform the POST request for the reservation.

5.3 Date and time

The date and time is stored in UTC timezone in the database of the server. The user gives the date and time in the local timezone, namely in the timezone where the mobile phone is located. Then, the user's date and time is tranformed , by the mobile application, in the UTC timezone so as that can be compared with the date and time of each reservation which is in UTC timezone.

It should be noted that whichever date and time is seen by the user, is displayed at the local time zone of the mobile phone. In this way, the user does not need to get confused with the time zones. He just enters and sees everything in his local time zone.

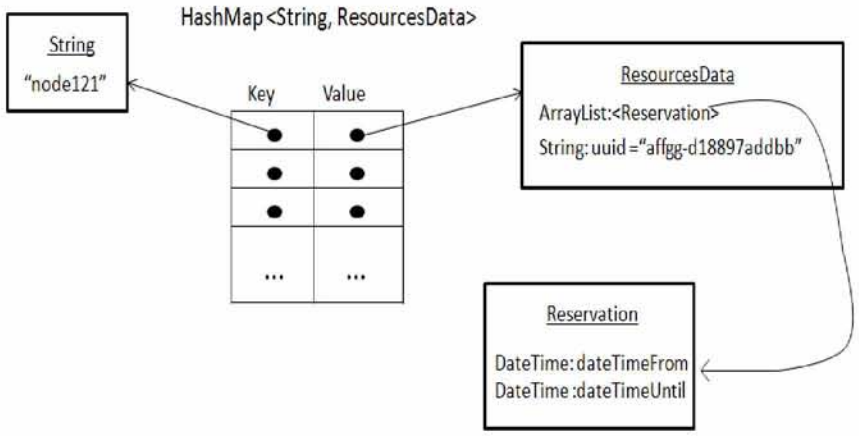

FIGURE 5.2: HashMap data structure storing all the resources

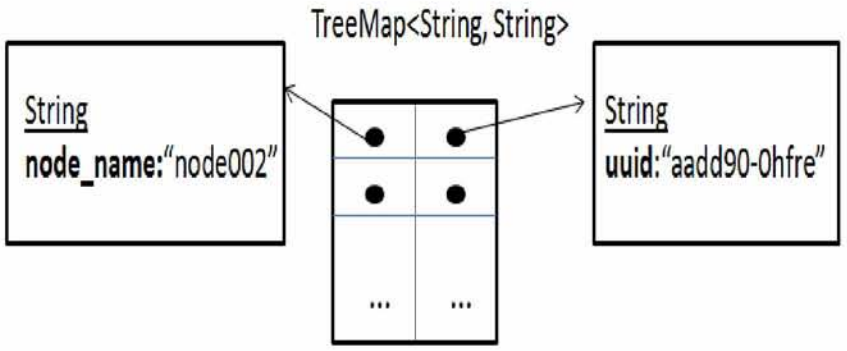

FIGURE 5.3: TreeMap data structure storing all the available resources

The SchedulerChooserFragment displays the local time at the top of the screen and the MyReservationsActivity displays each resources's reservation in the local timezone.

We have used the Joda-time library[4]for the date and time since overweights the standard JDK date/time classes. Joda-Time has been carefully designed to overcome all the issues the standard JDK date/time classes are blamed for. The standard JDK Calendar makes accessing "normal" dates difficult due to the lack of simple methods, while Joda-Time has straightforward field accessors such as getYear() or getDayOfWeek(). Joda library cares about the DayLight Saving Time, and adjusts the date and times accordingly. Also, it is easy to convert date and time from one timezone to another. For instance, this is an example how to convert a date and time from Europe/Athens timezone to UTC.

```
DateTimeZone zoneUTC = DateTimeZone.UTC;
DateTime dt = new DateTime("2004-12-13T21:39:45.618+03:00", zoneUTC);
```

5.4 Nested Fragments

Nested fragments[1] are released with the android 4.2. Nested fragments means that you can embed fragment in another. This is useful for a variety of situations in which you want to place dynamic and re-usable UI components into a UI component that is itself dynamic and re-usable. To nest a fragment, simply call getChildFragmentManager() on the Fragment in which you want to add a fragment. NitosTools application has as nested fragments the CheckResourcesFragment inside the AvailableResourcesFragment.

5.4.1 Bug in nested Fragments

However, as we were navigating across the tabs[2], the application crashed with IllegalStateException. This is a bug for nested fragments. Basically, the child FragmentManager ends up with a broken internal state when it is detached from the activity. The solution is to add the following to onDetach() of every Fragment which we call getChildFragmentManager() on:

```
@Override
public void onDetach() {
    super.onDetach();

    try {
        Field childFragmentManager = Fragment.class.getDeclaredField("mChildFragmentManager");
        childFragmentManager.setAccessible(true);
        childFragmentManager.set(this, null);

    } catch (NoSuchFieldException e) {
        throw new RuntimeException(e);
    } catch (IllegalAccessException e) {
        throw new RuntimeException(e);
    }
}
```

5.5 Platforms and Development Software

NitosTools was developed on a personal computer running Ubuntu 13.04 64-bit Linux operating system, where the following open source development tools were used:

- Eclipse Kepler IDE
- Open JDK 1.7.0
- Android SDK
- Android 4.3 platform(Jelly Bean)

Also, the following APIs and plug-ins for the Eclipse IDE were used:

- Android Development Tools (ADT)
- an Android Virtual Device (AVD)

The Joda library 2.3 was used for dates and times and is inserted in eclipse in this way: Create libs folder, then copy paste jode-2.3.jar in libs(create folder libs if not existed), and then right click and "Add to Build Path" .

Chapter 6

User Guide

In this Chapter we are going to present a user manual, explaining how UI interacts with the user.

6.1 Application's Launching

When the user launches the NitosTools Application., he/she is asked to enter the username and the password to login. The launcher icon is depicted in Figure 6.1.

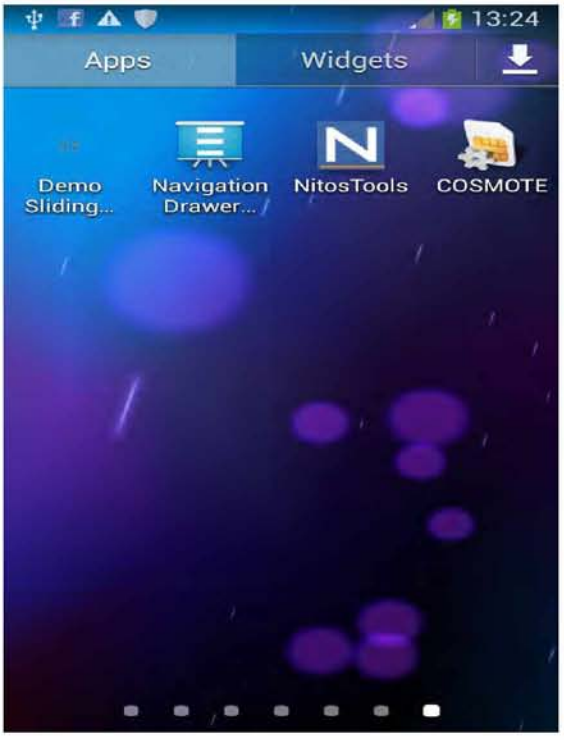

FIGURE 6.1: NitosTools Launcher Icon

6.2 How to Login

After the user launches the application, the LoginScreenActivity is shown.Then, when the user presses one of the editable fields, a keyboard with letters is shown.

If the user does not want to fill the username and password fields, everytime he/she launches the application, he/she should check the "Keep me signed in checkbox". In this way, eveytime the user launches the application, he/she sees the main activity screen. If the user wants to logout, he/she can logout pressing the logout option of the action bar's menu. The logout procedure is shown in Figure 6.2.

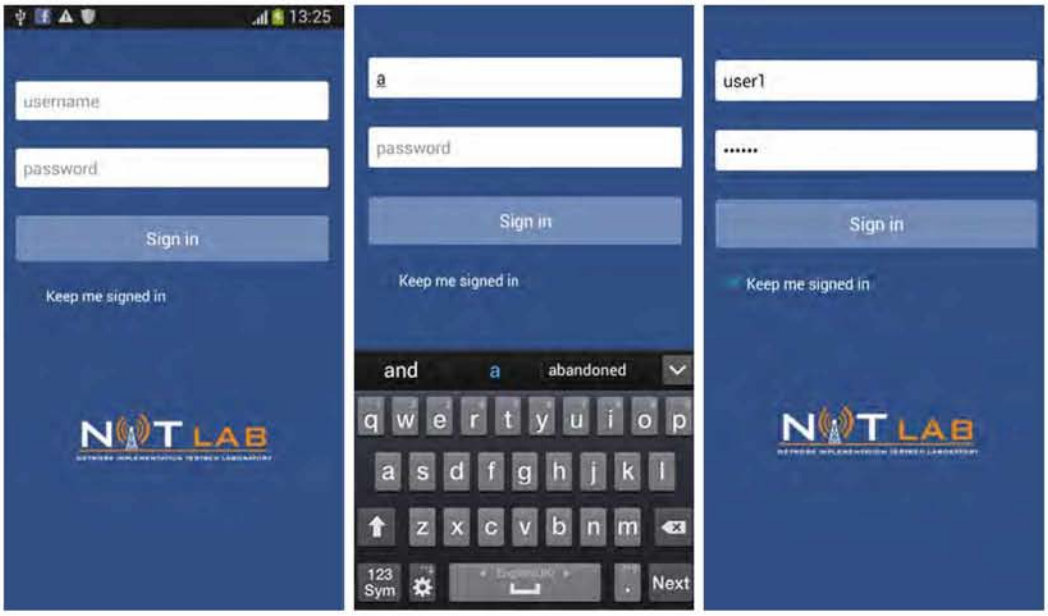

FIGURE 6.2: How to login to NitosTools activity

6.3 Main menu

After the user has logged in, he/she is redirected to the main menu screen. The main menu screen shows all the testbed tools and it is depicted in Figure 6.3. Right now, only the nitos scheduler item redirects to the Nitos Scheduler's screen. All the other testbed tools are unimplemented.

FIGURE 6.3: Main menu screen

6.4 How to make a reservation

When the user presses the Nitos Scheduler list item, he/she is redirected to the Nitos Scheduler's screen. The user can select the slice, the date, the time, and the duration of the reservation. Right now, the slice's selection is not implemented. The screen intended for the selection is shown in Figure 6.4. The selection procedure screen are shown at Figure 6.5. When the user is ready, he/she can press the "Check Available Resources" button, in order to be redirected to a screen displaying all available resources.

FIGURE 6.4: "Choose reservation's parameters" screen

If the user selects a date and time which are before the current date and time, and then presses the "Check Available Resources" button, a warning message "Date and

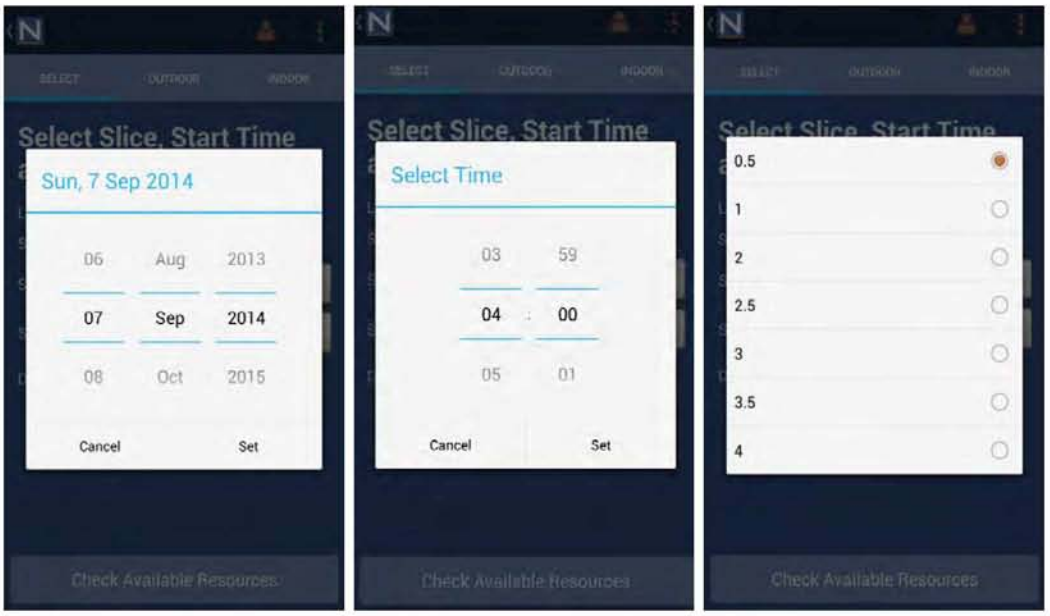

FIGURE 6.5: Select date, time, and duration screens

Time is before now" is shown. There is no meaning to check the resources' availability for a past date and time. Also, if the user presses the "Check Available Resources" button with no internet connection, a warning message "no network connection" is shown. The warning messages are shown in Figure 6.6.

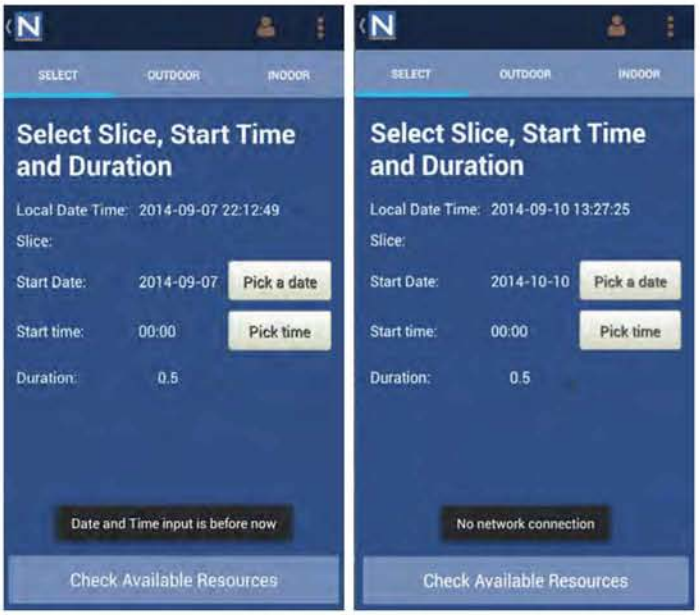

FIGURE 6.6: Warning messages in selection screen

After the user presses the "Check Available Resources" button with a valid date and time, and with network connection, a message "Please Wait" is shown. The connection to the server and the retrieval of the data takes some time, thus the waiting message is shown to the user. The waiting screen is shown in Figure 6.7.

FIGURE 6.7: "Waiting message in selection's screen

Then, the user is redirected to a screen displaying all the available resources. Screen displaying all the available resources is depicted in Figure 6.8. There is a list with blue color on the left of the screen where you can choose the category of the resources. Every time, the category of resources is clicked, it is colored with light blue. Each time the user presses a list item, a list of available resources, in form of checkboxes, appears on the right of the screen.

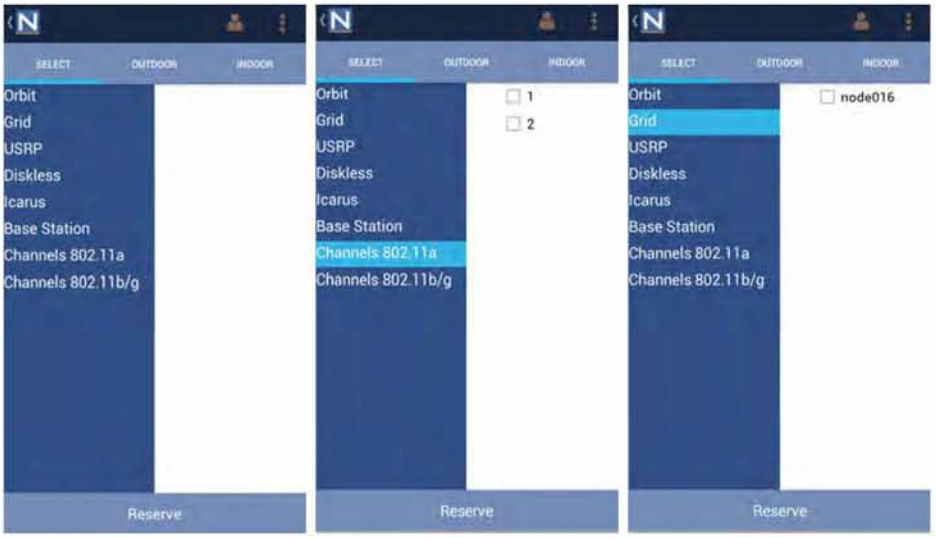

FIGURE 6.8: Screen displaying all the available resources

The user can select the resources that wants to reserve by checking the checkboxes. The checkboxes' state is not lost, while the user navigates through the list items. The selection of checkboxes is shown in Figure 6.9.

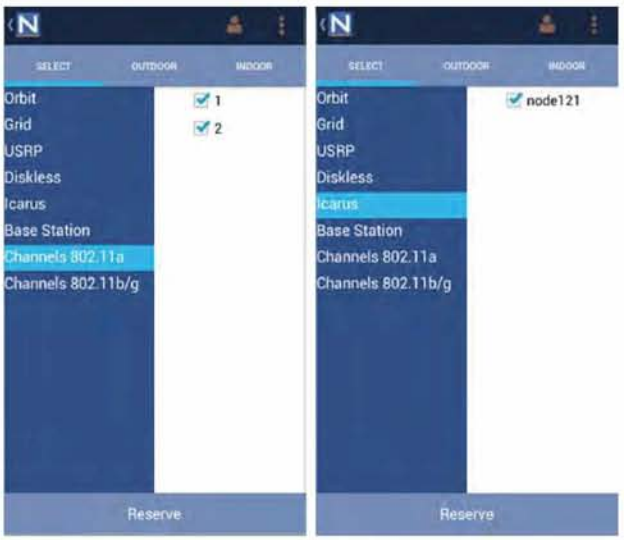

FIGURE 6.9: Selecting the resources

After the user has selected all the resources wants, he/she should press the Reserve button. After the clicking of this button, the user will be ridirected to a screen displaying a message about the successfullness of the reservation. This screen is displayed in Figure 6.10.

FIGURE 6.10: Message after the reservation

The location of the resources in the outdoor and indoor testbed is shown to the screen, selecting the second and third tab respectively. The screens displaying these testbeds are depicted in Figure 6.11.

FIGURE 6.11: Screens displaying the outdoor and indoor testbed

6.5 How to check the reservation status

If the user wants to see his/her reservation status, namely which reservation have been made, he/she should press the action bar's person's icon. When the user presses this button, a waiting "Please wait" message is displayed, since the application needs to retrieve data from the server. After the user presses this icon, he/she is redirected to a screen showing all the reservations. More specifically, the date and time about the start and the end of the reservations are depicted. The screens displaying the procedure for showing the reservation status are depicted .

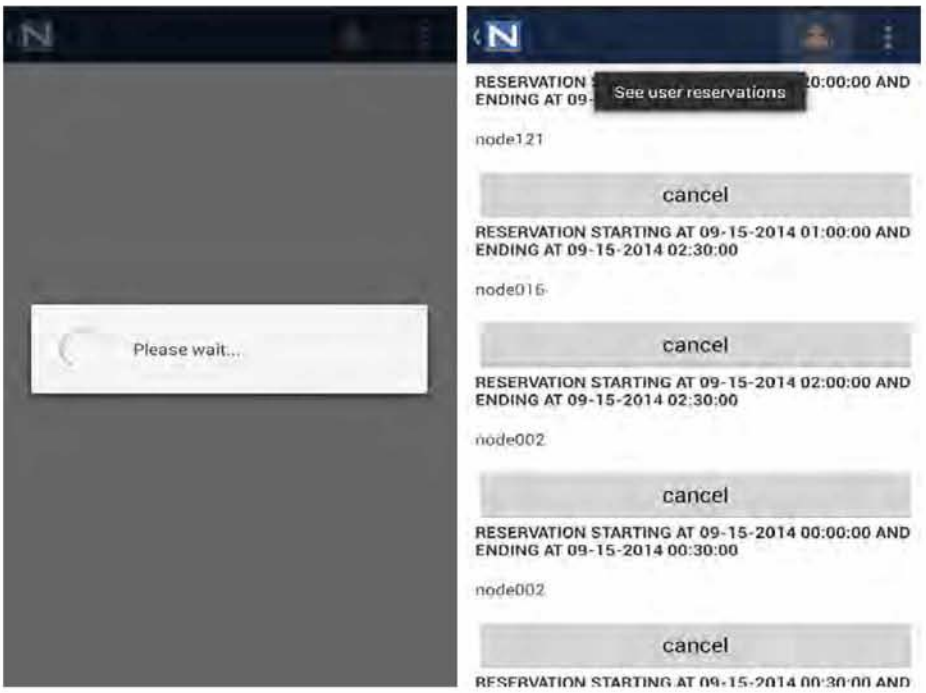

FIGURE 6.12: Reservation status of the user

6.6 How to log out

If the user wants to log out of the application, he/she can press the logout option of the action bar's menu. The logout option is depicted in Figure 6.13.

FIGURE 6.13: How to Log out

Chapter 7

Conclusion

In this final section the functionalities of NitosTools are restated and potential future work is discussed.

7.1 Conclusion

Within this thesis the Android Application NitosTools was implemented. In particular, an Android application was implemented that is responsible for managing the NITOS testbed resources. It guides the user through the reservation process, making sure that he does not make a reservation conflicting with reservations made by other users. The android application enables its user to select the parameters for the reservation, check the available resources, select resources, and reserve them. Also, the user can check his/her reservation status through NitosTools.

7.2 Future Work

An enhancement of this work could be the design of the user interface so as to work in landscape mode. Also, only the Nitos Scheduler is implemented from the testbed tools, the next step is the implementation of the rest as an Android Application.

Appendix A

UML Diagrams

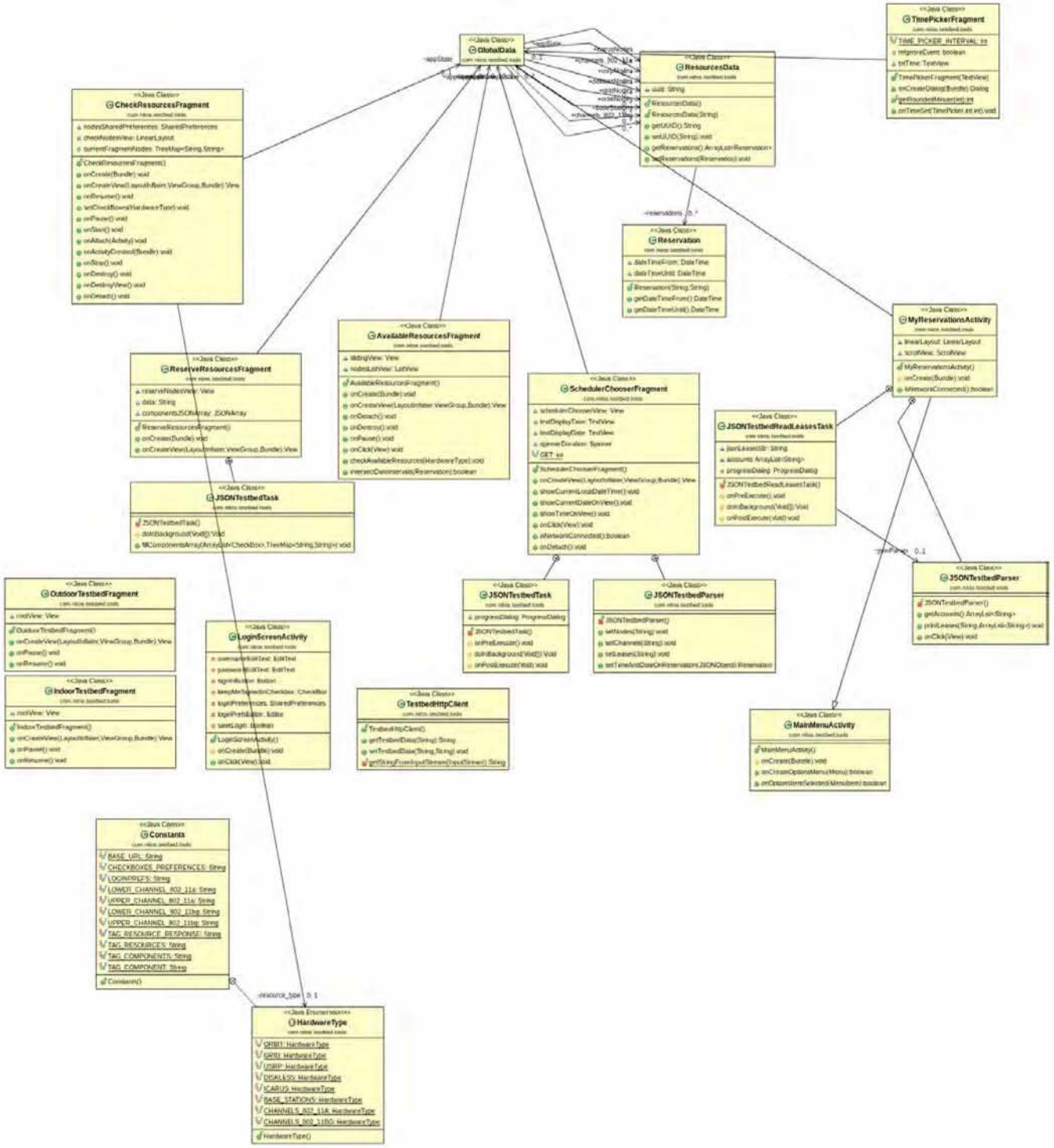

FIGURE A.1: NitosTools Application

Bibliography

[1] *http://developer.android.com/guide/index.html*

[2] *http://stackoverflow.com/*

[3] *http://nitlab.inf.uth.gr/NITlab/index.php/testbed-tools/nitos-scheduler*

[4] *http://www.joda.org/*

[5] *https://source.android.com/source/building.html*

[6] *https://source.android.com/source/initializing.html*

[7] Kok-Kiong Yap *Making Use of All the Networks Around Us:A Case Study in Android* CellNet'12, August 13, 2012, Helsinki, Finland, Stanford University.

[8] *http://www.stanford.edu/ huangty/cross_compile_patches.tar.gz*

[9] *http://wiki.cyanogenmod.org/w/Build_for_supersonic*

[10] *http://www.android-x86.org/documents/customizekernel*

[11] *http://comments.gmane.org/gmane.network.openvswitch.devel/21601*

[12] *http://www.slideshare.net/janghoonsim/virtualized-network-with-openv-switch*