International Journal of Advance Foundation and Research in Computer (IJAFRC)
Volume 2, Issue 5, May - 2015. ISSN 2348 – 4853

Disk Less Android Device

Abhijeet Daga[1], Manish Ghumnani[2], Prasanna Pawar[3]
Dept. of Computer Engineering, Vishwakarma Institute of Information & Technology
Savitribai Phule Pune University [1, 2, 3]

ABSTRACT-

Enabling disk-less/cheap Android devices (i.e. with minimal internal disk or flash memory). By minimal disk it means, your Android phone would have limited, say 2GB disk as opposed to 8/16/32GB internal storage and no external SD card slots. Many devices already don't have an external SD card slot (e.g. Google Nexus devices, Apple iPads etc.). The intellect, as Google explained, is that it is disconcerting for users. Apple/Google is trying to motivate users to Cloud. Also Tablets/Phones with larger internal flash memory cost more.

Keywords- NFS (Network File System), DVM (Dalvik Virtual Machine), Rootfs (Root File System), JDK (Java Development Kit), FUSE (File System in User space).

I. INTRODUCTION

1. Introduction

The number of Smart phone users and mobile application offerings are growing rapidly. A Smart phone is often looked to offer PC-like functionality, which requires powerful processors, abundant storage and long-lasting battery life. Nevertheless, their hardware today is still very limited and these limitations need to be taken into consideration. In an effort to alleviate the limitations of Smart phone storage, we are proposing cloud computing environment. Proposed system provides remote mounting of Android file system for disk-less/cheap devices. Not a single system is exploring the design of remote mounting of Android file system for disk-less/cheap devices. This will be advantageous for high battery and high privacy. Our system also has options by which the user can perform the selection for storage. Any user will have its need as it provides high privacy, battery, etc.

II. OVERVIEW OF THE SYSTEM

Through this project we propose an idea to load Android middle-ware (i.e. its root file system) directly from Cloud. We know that Dropbox enables cloud storage for user data/documents.[1] However, it is currently an application that user downloads from the app Store. This task will require to add support for cloud storage in a file system - something similar to NFS, but highly efficient by enabling intelligent caching [7]. This file system will use local disk/SD card as cache for the disk in the cloud. If the data requested is not found in the local cache, then a stumble to the Cloud (over the network) is prepared to fetch that data using Wi-Fi/3G/4G LTE, whatever technology is available. The challenge is that each time a cloud network trip is made over 3G/4G, the cellular data service is used and the user gets charged by their service provider (e.g. Airtel, Reliance etc.). This also drains battery on the device. Then this file system must be extremely efficient, so as to save network trips by intelligently caching files locally.

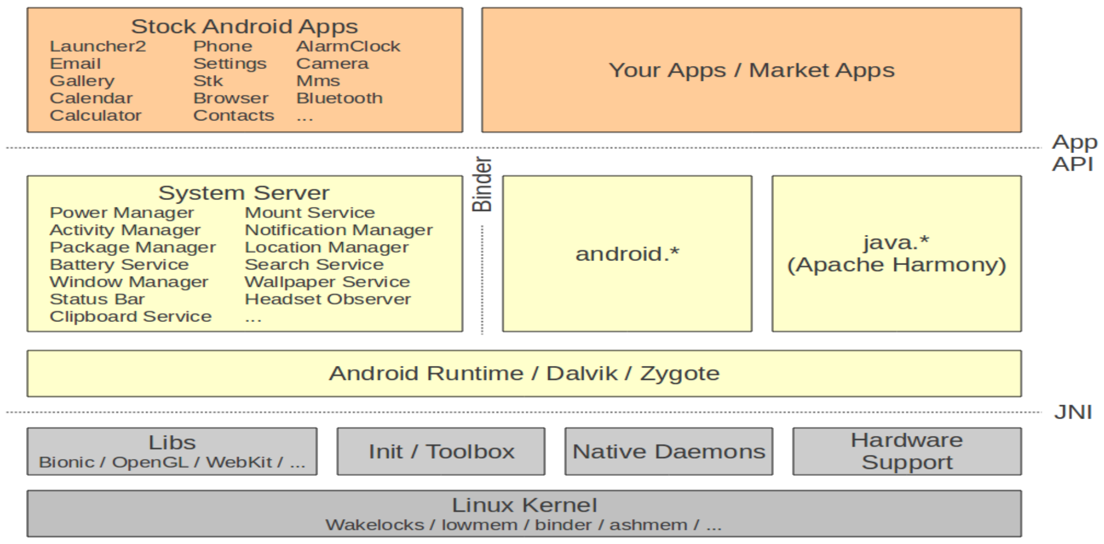


Fig.1 Android Internal Architecture [7]

2.1. System Architecture

Architecture defines an overall system.[2] Discovery module is responsible for carrying out discovery of devices connected to the network. When discovery is completed, the results are stored in discovery database. This is secondary database which is compared with actual database for finding the discrepancies. Reconciliation module then takes those discrepancies & ask the user to reconcile. If the user performs the reconciliation, then changes are updated in the main database.

2.1.1. Linux kernel

At the bottom of the layers is Linux - Linux 2.6 with approximately 115 patches. This provides basic system functionality like process management, memory management, device management like camera, keypad, display, etc. Also, the kernel handles all the things that Linux is really good at, such as networking and a vast array of device drivers, which take the pain out of interfacing to peripheral hardware.

2.1.2. Libraries

On top of Linux kernel there is a set of libraries including open-source Web browser engine Web Kit, well known library libc, SQLite database which is a useful repository for storage and sharing of application data, libraries to play and record audio and video, SSL libraries responsible for Internet security etc.

2.1.3. Android Runtime

This is the third section of the architecture and available in the second layer from the bottom.[6] This section provides a key component called Dalvik Virtual Machine which is a kind of Java Virtual Machine specially designed and optimized for Android. The Dalvik Virtual Machine makes use of the Linux core features like memory management and multi-threading, which is intrinsic in the Java language. The Dalvik Virtual Machine enables every Android application to run in its own process, with its own instance of the Dalvik virtual machine. The Android run time also provides a set of core libraries which enable Android application developers to write Android applications using standard Java programming language.

2.1.4. Application Framework

The Application Framework layer provides many higher-level services to applications in the form of Java classes. Application developers are allowed to make use of these services in their applications.

2.1.5. Applications

You will find all the Android application at the top layer. You will write your application to be installed on this layer only. Examples of such applications Contacts Books, Browser, Games.

2.1.6 Init

The user interacts with kernel at the time of booting of the device. Kernel gets loaded at the time of booting of the device and the process INIT gets called at startup. The kernel get aware about the partition since the rootfs is mounted over nfs [5]. Default file /root files will be available on the device only and supporting files will be available on cloud which will be faced at time of retrieving of files.

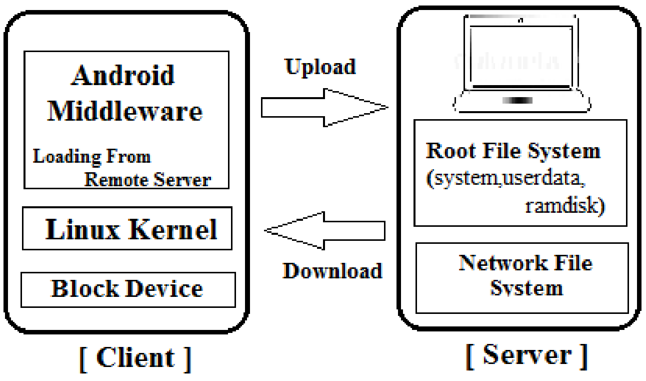


Fig 2. Proposed System

The above figure shows the interaction of the user through smart phone and further with cloud database.

2.1.7 Inter-process Communication

A special driver called "Binder" allows an efficient inter-process communication (I.PC)) which allows references to objects are passed between processes. The real objects are stored in Shared Memory. This way the communication between the processes is optimized as less data must be transferred.

2.1.8 Intelligent cache

The intelligent cache will keep an eye on supporting files which will be fetched from cloud and if the frequency of retrieving of same files is maximum, then they will remain in the cache and they will be fetched from the cache only in order to reduce the time required to fetch files from the cloud and also reduce the bandwidth and latency and try to avoid a trip to the cloud [8] [9].

2.1.9 Dalvik Virtual Machine

Android apps are commonly written in Java and compiled to byte code. They are then converted from JVM-compatible .class files to Dalvik compatible .dex (Dalvik Executable) files before installation on a device. The compact Dalvik Executable format is designed to be suitable for systems that are constrained in terms of memory and processor speed.

III. IMPLEMENTATION

The approach that we followed is to NFS-mount our root file system. This means that all files are actually on a host. The host is configured to export the file system, mount the file system at boot time and use it as its own. All files are visible on the host.

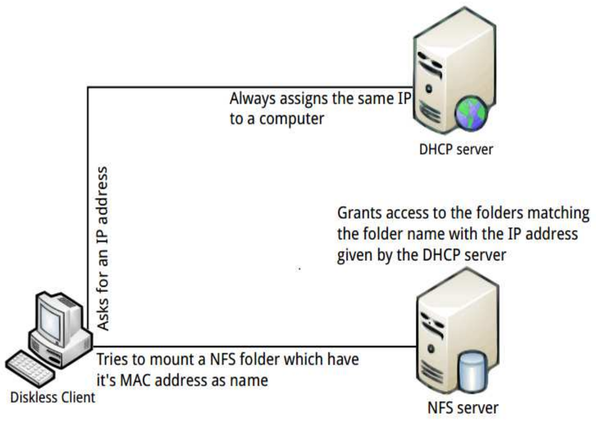

Always assigns the same IP to a computer
DHCP server
Asks for an IP address
Grants access to the folders matching the folder name with the IP address given by the DHCP server
Tries to mount a NFS folder which have it's MAC address as name
Diskless Client
NFS server

Fig 3. External Storage Mount [12].

As targets have increased in storage capacity and speed, the need for NFS-mounting root file systems will decrease. The technique is still useful when bringing up a new port or if you have lots of files that need to change together.

3.1 Building of kernel

Initially we started to build a custom kernel and gave it NFS and FUSE support. For the same we took the kernel source which is compatible with the device over which flashing of the kernel is to be carried out. And after the Z image is built from the kernel source, it is useful for creating the boot image file that will replace the one which is already there in the ROM of that device in order to flash the NFS and FUSE supported kernel.

NCURSES [13] is a clone of the original System. It is a freely distributable library and compatible with older version of curses interchangeably. It is a library of functions that manages an application's display on character-cell on the command line. In the remainder of the document, the terms curses are used

3.2 Root File system over network file system

To proceed with this we looked at how EMAC OE systems can be booted using NFS (Network File System) as the root file system. This method can be especially useful during development where the root file system is changing frequently. This can save time as well as wear on the on-board flash device.

3.2.1 Root File system

A complete root file system for the EMAC OE system [14] to boot from must be stored on the NFS server. The NFS server must be configured to allow clients to access this file system. The root file system does not have to be the directory shared by the NFS file system; it can be in a subdirectory, which means many root file systems can be shared by one NFS server.

3.2.2 Network File system

Steps for building of the kernel:

1. Installing the libraries ncurses

>sudo apt-get install libncurses5-dev

2. For Fetching the kernel source

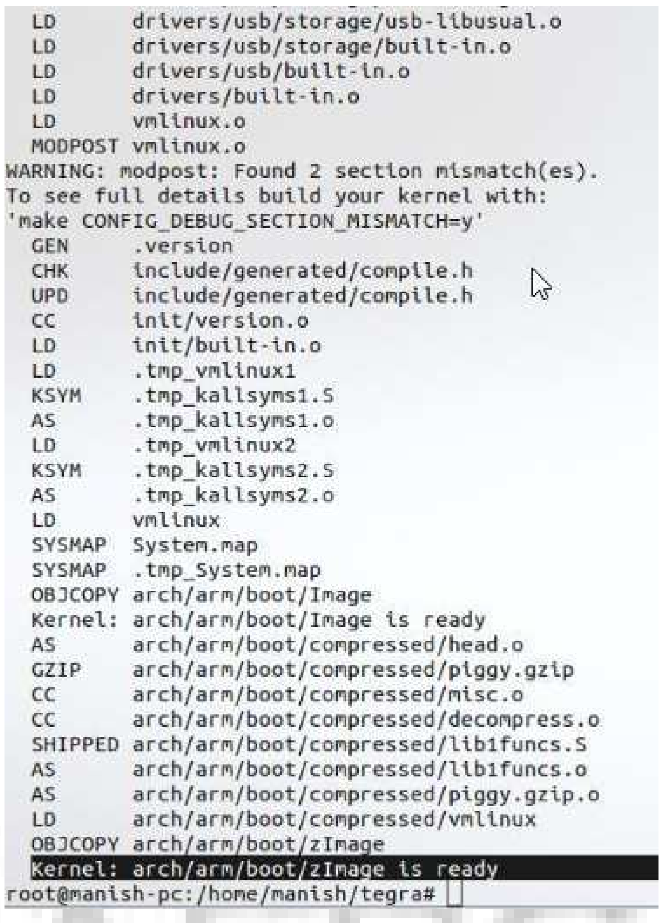
>gitclone https://android.googlesource.com/kernel/tegra kernel

3. For the building of kernel using cross compile tool chain

>makeARCH=armCross_Compile=arm-eabi-4.6/bin/arm-eabi

4. Building the source using "make"

>make



```
  LD      drivers/usb/storage/usb-libusual.o
  LD      drivers/usb/storage/built-in.o
  LD      drivers/usb/built-in.o
  LD      drivers/built-in.o
  LD      vmlinux.o
  MODPOST vmlinux.o
WARNING: modpost: Found 2 section mismatch(es).
To see full details build your kernel with:
'make CONFIG_DEBUG_SECTION_MISMATCH=y'
  GEN     .version
  CHK     include/generated/compile.h
  UPD     include/generated/compile.h
  CC      init/version.o
  LD      init/built-in.o
  LD      .tmp_vmlinux1
  KSYM    .tmp_kallsyms1.S
  AS      .tmp_kallsyms1.o
  LD      .tmp_vmlinux2
  KSYM    .tmp_kallsyms2.S
  AS      .tmp_kallsyms2.o
  LD      vmlinux
  SYSMAP  System.map
  SYSMAP  .tmp_System.map
  OBJCOPY arch/arm/boot/Image
  Kernel: arch/arm/boot/Image is ready
  AS      arch/arm/boot/compressed/head.o
  GZIP    arch/arm/boot/compressed/piggy.gzip
  CC      arch/arm/boot/compressed/misc.o
  CC      arch/arm/boot/compressed/decompress.o
  SHIPPED arch/arm/boot/compressed/lib1funcs.S
  AS      arch/arm/boot/compressed/lib1funcs.o
  AS      arch/arm/boot/compressed/piggy.gzip.o
  LD      arch/arm/boot/compressed/vmlinux
  OBJCOPY arch/arm/boot/zImage
  Kernel: arch/arm/boot/zImage is ready
root@manish-pc:/home/manish/tegra#
```

Fig. 4. Creating the zImage

To boot an EMAC OE system [15] over NFS, an NFS server must be available on the local network. This is often the same machine that is being used for software development. EMAC recommends using the **nfs-kernel-server** package available on most desktop Linux distributions, if setting up a new NFS server. Once the server has been installed, export a directory to use as the root file system. This is often done using the **/etc/exports** file. This document assumes that the root file system for the board will be located at **/srv/nfs/rootfs** on the NFS server.

3.2.3 Quick emulator

QEMU [16] is used for CPU emulation for user-level processes, allowing applications are compiled for one architecture to be run on another. Qemu is a virtual machine monitor: It emulates CPUs through dynamic binary translation and provides a set of device models, enable it to run a variety of unmodified guest operating systems. And used together with KVM in order to run virtual machines at near-native speed.

Steps for ROOTFS over NFS

1. Installing port map and NFS kernel server

> sudo apt-get install NFS-kernel-server port map (rebind)

2. Configure sharing folder in exports

> sudo gedit /etc/exports

>/tmp/nfs 10.0.2.15/255.255.255.0(rw,sync,no_root_squash ,insecure)

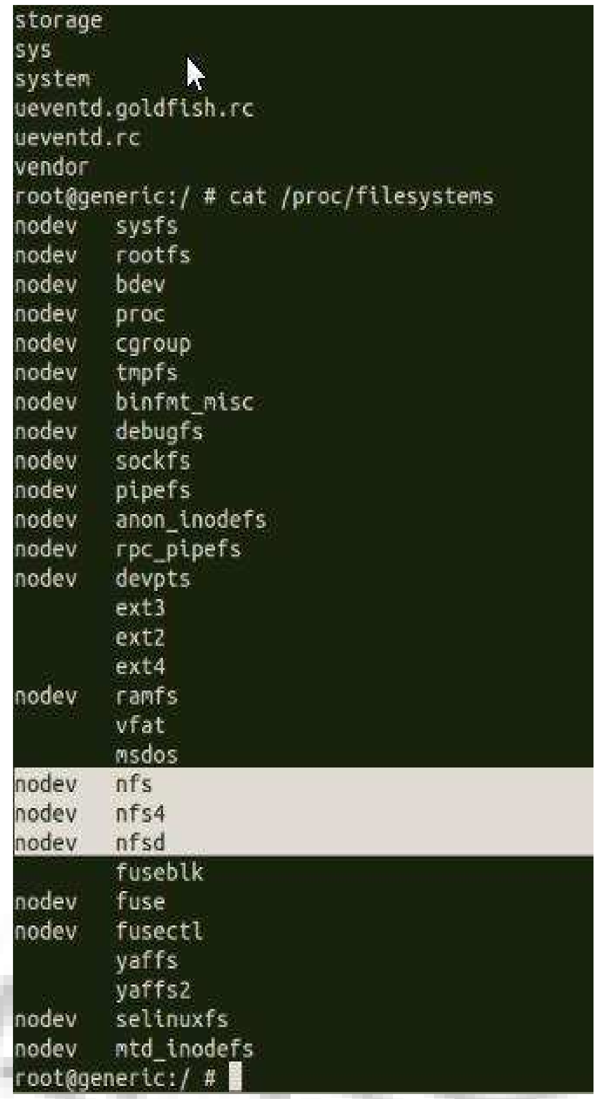
> /tmp/nfs 127.0.0.1(rw,sync,no_root_squash,insecure)

3. Kernel command line

>root=/dev/nfs

4.Starting the emulator

emulator-debuginit-kernel/home/manish/goldfish/arch/arm/boot/zImage -avd nx -qemu-M versatilepb -m 128M -append "root=/dev/nfs nfsroot=192.168.0.102:/srv/nfs/ rw"



```
storage
sys
system
ueventd.goldfish.rc
ueventd.rc
vendor
root@generic:/ # cat /proc/filesystems
nodev   sysfs
nodev   rootfs
nodev   bdev
nodev   proc
nodev   cgroup
nodev   tmpfs
nodev   binfmt_misc
nodev   debugfs
nodev   sockfs
nodev   pipefs
nodev   anon_inodefs
nodev   rpc_pipefs
nodev   devpts
        ext3
        ext2
        ext4
nodev   ramfs
        vfat
        msdos
nodev   nfs
nodev   nfs4
nodev   nfsd
        fuseblk
nodev   fuse
nodev   fusectl
        yaffs
        yaffs2
nodev   selinuxfs
nodev   mtd_inodefs
root@generic:/ #
```

Fig. 5. Kernel configured to support NFS and FUSE

3.2.4. Advantages of mounting rootfs over NFS:

- Easy to access and modify the rootfs.
- No limit no size.

3.2.5. Problems with NFS:

- It takes memory.
- The logs aren't kept after a reboot.
- Boot time is increased.
- Latency is observed in the system performance.

IV. FUSE BASED IMPLEMENTATION.

As we saw, there are issues with mounting of rootfs over NFS, which lead us to proceed with another approach, i.e. to try to create a file system that emulates NFS, providing ability to read and write to the server using FUSE based implementation. File system in User space (FUSE) [17] is an operating system mechanism for Unix-like computer operating systems that lets non-privileged users create their own file systems without editing kernel code. This is achieved by running file system code in user space while the FUSE module provides only a "bridge" to the actual kernel interfaces.

FUSE is particularly useful for writing virtual file systems (VFS) [18]. Unlike traditional file systems that essentially save data to and retrieve data from disk, virtual file systems do not actually store data themselves. They act as a view or a translation of an existing file system or storage device. In principle, any resource available to a FUSE implementation can be exported as a file system. File system is a core component of a functional operating system. Traditional File system development has been confined to the kernel space.

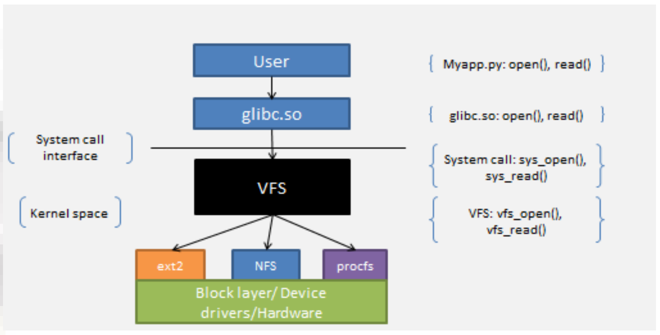


Fig 6. Virtual file system [19]

A customized, purpose-built, and user-driven File system development involves extensive knowledge of kernel internals, tools and processes. Alternatively, a user-space File systems are preferred over the kernel space File system, for ease of development, portability and developing prototypes file systems [20], particularly for intuitive abstraction of "non-file" objects. The main intention behind this is to create the file system using fuse that will work as network file system which is read/writes to remote server.

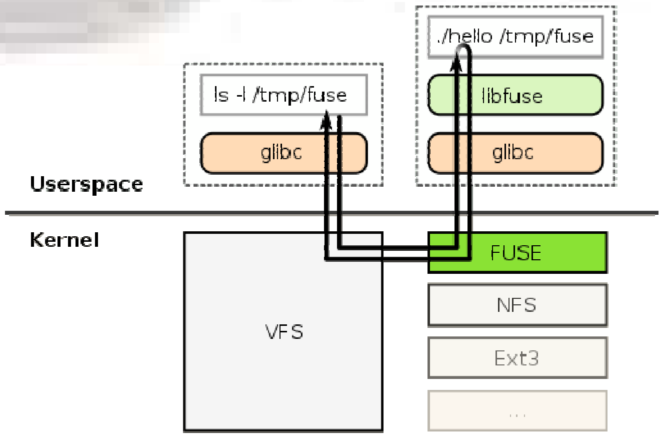


Fig 7. Working of File system in User space

4.1. Working of FUSE

With the help of FUSE [21] it is possible to implement a fully functional file system in a user space program. The fuse includes

> Library

> Simple installation

> Secure implementation o User space kernel interface.

> Usable by non-privileged users

> Runs on Linux kernels 2.4.X and 2.6.X

> Very stable over time.

The communication between the fuse kernel module and fuse library is done via a special file descriptor which is obtained by opening **/dev/fuse.** This file can be opened multiple times, and the obtained file descriptor is passed to the mount sys call, to match up the descriptor with the mounted filesystem. In UNIX kernel, a filesystem implementation is abstracted with virtual file system (VFS). VFS is acting as an interface to all available mounted filesystem on the computer file system. Fuse is a kind of a loadable kernel module and act as a file system to VFS. It registers itself with the VFS and opens a special device "/dev/fuse". The use space library "libfuse.so" polls the device, and read "/dev/fuse" device. FUSE module is an interface between the fuse device and the VFS. It receives file I/O requests from the VFS and writes those requests to "/dev/fuse" device.

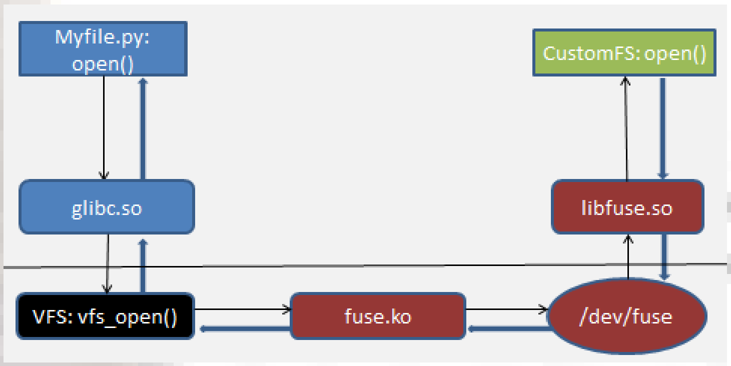


Fig.8 Block diagram of Fuse internal [22]

4.2 Cryptonite using Fuse

Cryptonite [23] is a framework used for providing mainly industry-standard encryption to not just mobile device's local data, but also the data which is on drop box storage. Cryptonite is an open-source android file encryption and decryption tool based on EncFS and TrueCrypt and used for open EncFS-encrypted files and folders stored on your mobile device as well as browse and export drop-box storage.

If the support of FUSE and NFS is provided to own kernel, then mounting of EncFS and TrueCrypt on our own volume supports cryptonite. Drop box allows us to encrypt and decrypt data present over the drop-box storage. Same we are doing building of filesystem which will have the access to upload and download (i.e. read and write data) so for this input will be the users' android data and server will be the drop box. And main intention behind this is we need to replace our own filesystem with EncFS so that our file system will load directly from the cloud storage. Replacing of filesystem with EncFS to read data from remote server so that our own filesystem will talk to remote server to upload and download data from drop-box which very much similar to NFS.

Cryptonite uses fuse to provide encryption for data which is on remote server as well as local to the devices. So that same we are going to replace it with our own filesystem which is built with the help of lib fuse filesystem.

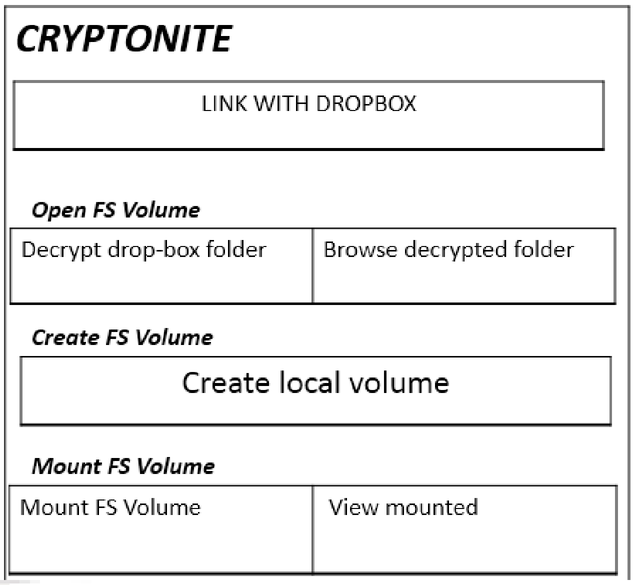


Fig.9 Block Diagram cryptonite

In cryptonite Files need to download and then re-upload to get them synchronized with encrypted drop box folder because drop-box doesn't support direct file synchronization on Android. Second, most Android devices don't support FUSE so that EncFS volumes can't be mounted for direct and transparent file encryption and decryption.

As a workaround, you can use one of the many apps that provide online storage synchronization (e.g. Folder Sync Lite or Drop sync). Setting up a scheduled sync to a local folder and mounting this local folder with Cryptonite will give you something very close to transparent file encryption and decryption.

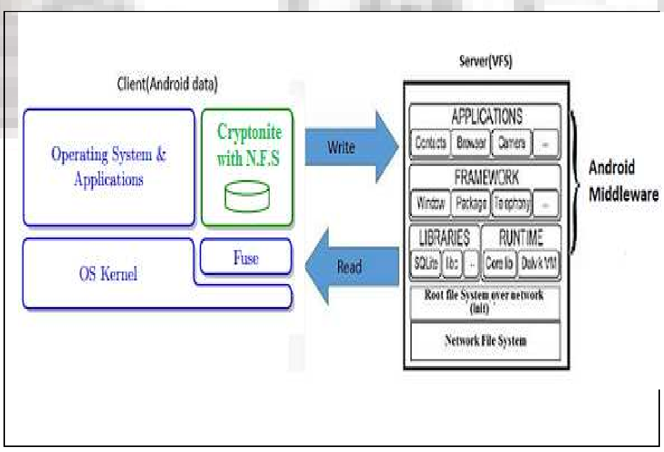


Fig.10 Read, write Architecture of Filesystem.

V. CONCLUSION

Using the concept of exploring the design mounting of android file system, i.e. loading the android middle-ware from the cloud. By observing various characteristics such as latency, battery consumption a cost model can be prepared. Once loaded into device memory and cached on the device, it won't consume more data than what apps anyway do by communicating with their in-cloud servers.

VI. RELATED WORK

1. Cloud based Mobile Augmentation - offload some pieces of work in the cloud.

a. Clone cloud - partitions applications to identify what could be offloaded to cloud. Looks for CPU-intensive pieces in a process (e.g. a tight loop performing encryption). Uses compiler/Dalvik virtual machine to identify what could be independently offloaded. There are limitations [24], [25].

b. MAUI - offloads stuff to the Cloud to save energy/battery [26].

c. COMET - migrates apps to Cloud [27].

2. Cloudlets - offloads Virtual Machines (VMs) to a nearby server called Cloudlet or to a distant server for offloading to save energy and increase performance[28].

3. Thin client approach - it takes a VNC based approach to run Android in the Cloud and forward screen output to the device

a. "Smartphone over IP" - this also sends sensor data from device to Cloud

4. Other approaches

"Diskless Android System" - proposes to use Android on old Desktop machines with mediocre CPU, limited memory, and no disk.

None of the above ideas is exploring the design of remote mounting of Android file system for diskless/cheap devices. The issues (as explained above) that will be addressed in this concept (privacy, battery, latency, etc.) could potentially make a research thesis.

VII. FUTURE WORK

After the loading of middle-ware from the cloud, part of device memory can be used as cache for data fetched from the server. It will consist of those libraries, Application ProgrammingInterfaces (Core Java Libraries, Dalvik virtual machine) of recently used applications and applications that are used frequently. So that any further fetch will be directly made from the cache part of the device and hence it will reduce the trip over cloud ultimately reducing the latency and battery consumption.

Also, an approach of using Fuse to emulate the Network File system(NFS) can be employed. Ability to Code in User Space without worrying about the complexity of kernel programming stands it in good stead.

VIII. REFERENCES

[1] "Dropbox - Home - Online backup, file sync and sharing made easy." http://www.dropbox.com/.

[2] "Android Developers," http://developer.android.com/index.html.

[3] Sooman Jeong1, Kisung Lee 2, and Youjip Won I/O Stack Optimization for Smartphone, USENIX annual Technical Conference 2013.

[4] Eric Y. Chen and Mistutaka Itoh,Virtual smartphone over IP, NTT Information Sharing Platform Laboratories, Tokyo 180-8585, Japan 2011.

[5] http://developer.android.com/tools/help/emulator.html

[6] https://www.digitalocean.com/community/tutorials/how-to-set-up-an-nfs-mount-on-ubuntu-12-04

[7] http://elinux.org/Android_Architecture

[8] Byung-Gon Chun and Petros Maniatis,Cloud based Mobile Augmentation, Intel Research Berkeley 2010.

[9] Eduardo Cuervo and Aruna Balasubramanian MAUI - offloads stuff to the Cloud to save energy/battery, University of Massachusetts Amherst.

[10] Mark S. Gordon and D. Anoushe Jamshidi, COMET - migrates apps to Cloud, USA, 2012.

[11]. oumya Simanta and Kiryonh Ha, Cloudlets: Mobile Code Offload in, Hostile Environments, USA, 2007

[12]. Allan Pierra Fuentes, Miguel Albalat Aguila and Cealys Alvarez Trujillo, Android Diskless System

[13].https://tldp.org/HOWTO/NCURSES-Programming- HOWTO/intro.html

[14], [15] http://wiki.emacinc.com/wiki/Booting_with_an_NFS_Root_Filesystem.

[16] https://en.wikipedia.org/wiki/QEMU

[17], [18], [21] https://en.wikipedia.org/wiki/Filesystem_in_Userspace

[19], [20] Vishal Kanaujia and Chetan Giridhar, FUSE'ing Python for Development of Storage Efficient File System, 244-941-1-PB

[21], [22] https://fuse.sourceforge.net/

[23]https://code.google.com/p/cryptonite/

[24] https://www.usenix.org/legacy/event/hotos09/tech/full_papers/chun/chun_html/index.html

[25] http://eurosys2011.cs.uni-salzburg.at/pdf/eurosys2011-chun.pdf

[26] http://www.cs.duke.edu/~ecuervo/downloads/maui.pdf

[27] https://www.usenix.org/system/files/conference/osdi12/osdi12-final-11.pdf

[28] http://elijah.cs.cmu.edu/DOCS/cloudlet_hostile_MobiCASE2012_camera_ready.pdf

Authors

Abhijeet S. Daga currently pursuing B.E. in Computer Engineering

At Vishwakarma Institute of Information Technology, Savitribai
Phule Pune University. His areas of interest are cloud computing,
Kernel development.
He is currently working on project of Disk less Android Device.

Prasanna T. Pawar currently pursuing B.E. in Computer Engineering
At Vishwakarma Institute of Information Technology, Savitribai
Phule Pune University.His areas of interest are System Software
And Application Programming.
He is currently working on project of Disk less Android Device.

Manish R. Ghumnani is currently pursuing B.E in Computer Engineering
At Vishwakarma Institute of Information Technology, Savitribai
Phule Pune University. His areas of interest are
Systems Programming and Operating System. He is currently working
On the project - Diskless Android Device.