

Exploring the Potentials and Horizons of Network Block Device (NBD) Integration on the Android Ecosystem

by

Tanvir Rahman

22241134

Tanjid Ahmed

22341065

Md. Sakibur Rahman

22341071

Amreen Hossain

19101505

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science

Department of Computer Science and Engineering
School of Data and Sciences
Brac University
September 2023

© 2023. Brac University
All rights reserved.

Declaration

It is hereby declared that

1. The thesis submitted is our original work while completing the degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material that has been accepted or submitted for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

Student's Full Name & Signature:

Tanvir

Tanvir Rahman
22241134

তানজিদ

Tanjid Ahmed
22341065

Md. Sakibur Rahman

Md. Sakibur Rahman
22341071

Amreen Hossain

Amreen Hossain
19101505

Approval

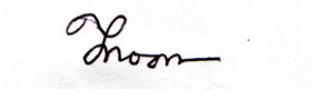
The thesis titled “Exploring the Potentials and Horizons of Network Block Device (NBD) Integration on the Android Ecosystem” submitted by

1. Tanvir Rahman (22241134)
2. Tanjid Ahmed (22341065)
3. Md. Sakibur Rahman (22341071)
4. Amreen Hossain (19101505)

of Summer 2023 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on September 21, 2023.

Examining Committee:

Supervisor:
(Member)



Dr. Jannatun Noor
Assistant Professor
Department of Computer Science and Engineering
Brac University

Thesis Coordinator:
(Member)

Golam Rabiul Alam
Professor
Department of Computer Science and Engineering
Brac University

Head of Department:
(Chair)

Sadia Hamid Kazi
Chairperson and Associate Professor
Department of Computer Science and Engineering
Brac University

Ethics Statement

This research adheres to the highest ethical standards and has been conducted by the guidelines set forth by BRAC University.

We confirm that the data collected for this research is authentic and accurately reflects the results of our study. All data was collected and analyzed rigorously and integrity, and any errors or discrepancies have been disclosed. We have also disclosed any conflicts of interest that may have influenced the interpretation of the data.

Abstract

The NBD (Network Block Device) protocol plays a pivotal role in enhancing the Android ecosystem, particularly in terms of storage management and optimizing power efficiency on ARM devices. By facilitating network block-level storage access, NBD enables Android devices to seamlessly connect to remote storage resources, expanding their storage capacity without needing physical upgrades. This is especially beneficial for handheld devices, where storage limitations can hinder functionality. Furthermore, NBD contributes to power efficiency by leveraging large data block write operations on cloud NBD servers. By minimizing the need for frequent data transfers and optimizing network communication, NBD reduces power consumption, extending Android devices' battery life and making them more energy-efficient and sustainable for users. Moreover, the NBD protocol facilitates the integration of Android devices with a wide range of hardware peripherals, enabling them to function as adaptable hubs for interacting with diverse IoT devices and sensors. The aforementioned feature enables an Android-operated portable device to transform into a resilient and versatile controller for Internet of Things (IoT) networks, smart home appliances, and various hardware components. This augmentation significantly amplifies its functionality and significance within the swiftly progressing realm of interconnected devices. However, integrating the NBD protocol inside the Android ecosystem is a prominent catalyst for enhancing efficiency, storage flexibility, and adaptability. Consequently, this integration contributes to ARM-based Android devices' increased capabilities and versatility.

Keywords: Network Block Device; Cloud Storage; Android Kernel; Linux; Network Attached Storage; Internet of Things.

Acknowledgement

This research was conducted at and rigorously supported by BRAC University.

To begin with, all praise to the Great Almighty for keeping us safe and healthy throughout the research period, for which we have finished our research timely.

We want to extend our heartfelt thanks to our supervisor, Dr. Jannatun Noor, for her expert guidance and for allowing us to work under her mentorship. We are truly grateful for her steadfast support and guidance throughout the research process.

Finally, we want to thank our parents for their support and prayers.

Table of Contents

Declaration	i
Approval	ii
Ethics Statement	iii
Abstract	iv
Acknowledgement	v
Table of Contents	vi
1 Introduction	2
1.1 Background	3
1.2 Motivation	5
1.3 Objectives & Contribution	6
1.4 Research Structure	7
2 Context & Overview	8
2.1 *nix Philosophy	8
2.1.1 File System	8
2.1.2 Symbolic Links	9
2.1.3 User Space and Kernel Space in Linux	9
2.2 Linux Kernel	10
2.2.1 Block Devices	10
2.2.2 Character Device	10
2.3 Loadable Kernel Modules (LKMs)	11
2.4 Android Kernel	12
2.5 Network Block Device (NBD)	13
2.5.1 Access Control	14
2.5.2 Network Security	14
2.5.3 Authentication	14
2.5.4 Encryption	14
2.5.5 NBD Server Configuration	15
2.6 NBD as Loadable Kernel Module (LKM)	15
2.7 Android SELinux Policy	15
2.8 Our Adhered SELinux Policies	16
2.9 Symbolic Linking Philosophy	17

2.10	Waydroid	18
3	Related Works	19
3.1	Background	19
3.2	Taxonomy of NBD, Linux and Android	20
3.2.1	Android Storage System Expansion	20
3.2.2	Storage I/O in Cloud Enviroment	22
3.2.3	Difference Between Linux and Android	25
3.2.4	Different Implementations of Network Block Device	27
3.2.5	Security And Data Privacy	29
3.2.6	Other Topics Regarding NBD, Linux and Android	29
4	Methodology & Security Architecture	32
4.1	Client-Server Architecture	33
4.1.1	Client	34
4.1.2	Server	35
4.2	Security Architecture	36
5	Implementation	40
5.1	System Specification	40
5.1.1	Server	40
5.1.2	Network Device	40
5.1.3	Client	40
5.2	Configuring the Server	41
5.2.1	Package Installation	41
5.2.2	NBD Server Configuration	41
5.2.3	Automatic Startup	42
5.2.4	Server Start	42
5.2.5	Server Status Verification	42
5.3	Configuring Client	43
5.3.1	Package Installation	43
5.3.2	Client Connection	43
5.3.3	Block Device Operations	43
5.3.4	Unmount and Disconnect	43
5.3.5	Waydroid	44
5.3.6	Symbolic Linking for Dynamic Partition	44
6	Experimental Evaluation	45
6.1	Configuration for Performance Test	45
6.1.1	Preparation	46
6.1.2	Bandwidth Test for Read Sectors	46
6.1.3	Bandwidth Test for Write Sectors	46
6.1.4	File System Test for Read FS	46
6.1.5	File System Test for Write FS	46
6.1.6	Mount and Random Access Test	47
6.1.7	Cleanup	47
6.2	Comparative Analysis	47
6.2.1	Storage Capabilities	48
6.2.2	Power Consumption	48

6.2.3	Cost Structure	51
7	Open Discussion & Challenges	53
7.1	Theoretical Comparative Study	53
7.2	Challenges	54
7.2.1	Hardware Dependency	54
7.2.2	Software Compatibility	55
7.2.3	Limited Comparison Candidates	55
7.2.4	Performance Metrics Selection	55
7.2.5	Network Environment Variability	55
7.2.6	Limited Generalizability	55
8	Conclusion & Future Prospects	56
8.1	Future Prospects	56
8.2	Conclusion	57
	Bibliography	61

Chapter 1

Introduction

The block device mechanism plays an important role in the input/output (I/O) system of the Linux operating system. Block devices are a classification of storage devices that operate at the block level, whereby data is accessed and altered in specified block sizes, typically 512 bytes or 4 KB. Block devices consist of a diverse range of storage media such as solid-state drives (SSDs), hard drives, USB drives and other similar devices. Block devices are integral components inside the Android ecosystem, vital to storage and data management. The predominant storage material employed by Android devices is NAND flash memory. Acknowledging that the aforementioned storage is organized and retrieved as block devices on the system level is vital. The block devices under consideration offer a layer of abstraction for the storage hardware they are connected to. This abstraction enables consistent read and write operations on blocks of a fixed size, typically measuring 4 KB. The maintenance and structure of data stored on block devices is overseen by the file system employed by Android, such as ext4 or F2FS. Block devices encompass several possibilities, including internal storage for applications and user data and external storage alternatives like microSD cards.

Furthermore, Android devices can connect with external block devices, such as USB drives or external SSDs. These devices are recognized and processed as block devices upon access. Adopting block devices in the Android ecosystem is crucial for enabling efficient data storage, retrieval and management procedures, substantially contributing to Android devices' overall performance and functionality.

The Network Block Device (NBD) is a feature of the Linux kernel that was deliberately designed and built to be very advanced. It allows users to view storage devices remotely on a block-by-block basis. By making a remote storage resource into a block device, this feature lets a Linux system install, access and use it like a local storage device. The NBD method is based on the client-server model, in which a server provides access to block devices. After that, clients can use these block devices from afar through a stable network link. It is very useful in storage systems and networks with many nodes. It makes it easier to share and combine info across different platforms. The NBD method makes storage management more flexible by letting you connect and disconnect remote block devices without any trouble. NBD is flexible because it can be used successfully in many situations, such as network storage, virtualization and cloud computing.

The NBD protocol and its technologies make sharing block devices like disc drives and partitions over a network easy. Also, NBD provides a flexible and optimized solution for remote storage access, empowering various applications such as virtualization, distributed computing and remote disk imaging. The key features will be elaborated upon in the following sections.

Furthermore, integrating the NBD into the Android operating system presents a significant opportunity to enhance storage capabilities. This feature allows Android devices, particularly those with limited internal storage capacity, to effectively connect with and utilize remote block-level storage resources. By establishing connections with remote servers or network-attached storage (NAS) devices, Android devices can leverage NBD to expand their capacity. It is particularly advantageous for Android users who heavily depend on their smartphones for scenarios like data-intensive activities such as video streaming or content creation. It helps seamless remote access and efficient management of large files. Additionally, NBD's effective data retrieval and management capabilities can optimize power efficiency by reducing the energy consumption linked to ongoing data synchronization. In summary, integrating Network Block Device (NBD) into the Android ecosystem promises a scalable and energy-efficient solution to address storage limitations. This has the potential to enhance the versatility and functionality of Android devices.

1.1 Background

Traditional data-at-rest mechanisms in the Android ecosystem involve storing data locally on the device's internal or external storage. This data is typically encrypted to protect sensitive information, ensuring it remains secure even if the device falls into the wrong hands. Android provides sophisticated encryption methods, such as File-Based Encryption (FBE) starting from Android version 7.0 API level 24 and Full Disc Encryption (FDE) starting from Android version 5.0 level 21 till Android version 7.0 level 24, that effectively protect stored data, including user files, app data and system data [33]. Users can set up encryption during device initialization or enable it through security settings, creating a secure environment for their data at rest. Additionally, Android provides APIs and guidelines for developers to implement further data protection within their apps, ensuring comprehensive security for data stored on Android devices.

Traditional data writing in the Android ecosystem consumes power and involves significant costs and resource utilization. When data is written to storage devices like NAND flash memory, power is required to program memory cells. This power consumption impacts the battery life of Android devices, which is a crucial consideration for users. Additionally, more extensive and faster storage options, such as high-capacity NAND chips or Embedded MultiMediaCard (eMMC), come at a premium, significantly impacting the overall cost of manufacturing Android devices. Several factors, especially the expanding size of mobile applications and games drive the demand for more significant data-writing capabilities. As game developers create more graphically advanced and immersive experiences, the file sizes of these games have grown significantly. To ensure seamless gameplay and app experiences, Android devices need to handle these large game installations and updates efficiently. Hence, the ability to write large volumes of data quickly and reliably is paramount

for delivering the best user experience. As of 12 September 2023, the price of the iPhone 15 128GB, 256GB and 512GB is respectively 799 USD, 899\$ USD and 1099\$ USD, which suggests a parabolic relationship between price and storage [35]. Thus, this increased demand for storage capacity and speed poses challenges regarding power efficiency and cost management, making it essential for Android device manufacturers and developers to strike a balance between storage capabilities, power consumption and affordability.

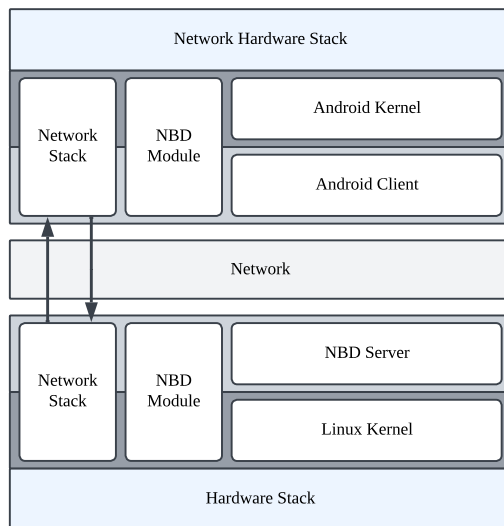


Figure 1.1: Overview

In this figure 1.1, we can see that a network-based approach can be implemented to address the challenges of extensive data writing, particularly for resource-intensive applications like games. Instead of storing these large game files directly on the Android device, games can be designed to write data commands locally, which are then efficiently processed and transferred to a remote server via a network connection. This server-side storage can accommodate significant game assets and updates, reducing the need for extensive local storage on Android devices. By leveraging this network-centric model, devices can prioritize efficient data processing over local storage, contributing to enhanced power efficiency and cost savings while still delivering high-quality gaming experiences. This approach aligns with the cloud gaming and content streaming trend, allowing users to enjoy resource-intensive content without needing massive local storage capacity.

The emergence of 5G networks is a significant milestone in the context of Android devices, especially concerning extensive data writing and energy efficiency. While local disk write speeds have improved over the years, 5G networks offer data transfer rates that can rival and sometimes even exceed, the speed of writing data to local storage [23]. This allows Android devices to offload extensive data writes to remote servers efficiently, reducing local power consumption during intensive data operations [28].

Network Block Device (NBD) can be a crucial beneficiary of this trend. By leveraging the high-speed and low-latency capabilities of 5G networks, NBD can enable Android devices to access and write data to remote block devices seamlessly. This

approach conserves energy and reduces the need for extensive local storage infrastructure, contributing to more energy-efficient and cost-effective Android devices.

The synergy between 5G networks, Android devices and NBD technology highlights the potential for a more energy-efficient and scalable approach to managing extensive data writes. This evolution aligns with the growing demand for resource-intensive applications and games, offering users an enhanced experience without compromising power efficiency or storage costs.

1.2 Motivation

Our motivation is to implement the Network Block Device (NBD) in the Android ecosystem, which is deeply rooted in the desire to explore unidentified features within the Android platform and to push the boundaries of what this versatile ecosystem can achieve. The inspiration for this study is derived from the significant advancements observed in network technologies, specifically the advent of 5G networks. We have demonstrated these networks' potential to facilitate rapid and efficient data transfer with minimal delay, potentially revolutionizing the handling of big data read/write on Android devices. The necessity for efficient data storage and management, given the ever-increasing size of applications and content, became evident. The idea that NBD, a well-established technology in the Linux realm, could be adapted for Android, thus bringing a new level of flexibility and efficiency to the Android storage ecosystem, served as a driving force. We aim to enable Android devices to access remote block-level storage resources quickly and easily, enabling users to expand their storage without physical restrictions and maximizing energy efficiency and cost-effectiveness.

Motivation was sustained throughout the research process by the intricate technical aspects of implementing NBD on a platform like Android, characterized by differences in architecture and system requirements. The prospect of enabling Android devices to access remote block-level storage resources efficiently remained a significant driving force and overcoming the technical challenges due to Android's non-development-friendly and tightly integrative commercial perspective was challenging and intellectually stimulating. Collaboration played an instrumental role in sustaining motivation, with the team's diverse skill sets and perspectives fostering an environment of creativity and problem-solving. Ideas were shared, approaches were debated and technical hurdles were overcome collectively. Furthermore, the enthusiasm of the Android community for innovation and the potential impact of the research fueled determination. It was recognized that the work could benefit users, developers and manufacturers seeking more efficient storage solutions for Android devices.

As the research progressed, tangible outcomes began to reinforce motivation. Witnessing the successful integration of NBD into the Android ecosystem, with the ability to access and utilize remote block devices efficiently, marked a gratifying milestone. It demonstrated the practical applications and benefits of the research efforts, further propelling the team forward.

However, the research journey involving the implementation of NBD in the Android ecosystem is a testament to the power of motivation, teamwork and the pursuit of

technological advancement. Inspiration was drawn from the research’s possibilities and its potential for reshaping how storage and data are managed on Android devices. The collective motivation remains unwavering as new horizons continue to be explored, contributing to the evolution of the Android ecosystem.

1.3 Objectives & Contribution

Implementing a Network Block Device into the Android ecosystem involves a range of technically sophisticated objectives. The primary objective is to optimize and empower Android devices with the essential scalability and adaptability required to excel in today’s dynamic digital landscape. These objectives are aligned synergistically to optimize the platform’s storage and energy efficiency, thereby expanding the capabilities of Android as a robust command center for the Internet of Things (IoT). Android smartphones leverage Network-Based Data (NBD) to streamline data centralization and empower seamless remote accessibility.

Utilizing cloud infrastructure enables and facilitates enhanced content management efficiency. This cost reduction and enhanced storage scalability make Android devices adaptable to dynamic data requirements. Integrating NBD in Android elevates its capabilities in efficiently navigating the intricacies of the modern digital landscape, merging technical prowess with practical utility.

- **Storage Efficiency:** Enhance storage efficiency by enabling Android devices to seamlessly access remote block-level storage, catering to the expanding dimensions of applications and media.
- **Power Efficiency:** Optimize power efficiency by leveraging remote server infrastructure for offloading storage operations, thereby effectively conserving battery life.
- **IoT Control Hub:** Empower Android devices to function as IoT control hubs through the seamless attachment and management of block-level devices via network connectivity.
- **Data Centralization:** Facilitate seamless cloud-based storage for centralized data, empowering users to retrieve their data from any location conveniently.
- **Cost-Effective Storage:** Enhance cost-effectiveness by leveraging remote storage resources, enhancing Android device’s affordability.
- **Scalability and Flexibility:** Enhance the adaptability of Android devices by providing scalable and flexible storage solutions to cater to a wide range of use cases.

The experimental implementation of Network Block Device (NBD) through Waydroid on Ubuntu 23.04 presents a valuable contribution to the ecosystem. It showcases the potential of leveraging NBD technology within the Android environment. It offers a glimpse into a future where Android devices can seamlessly access and manage remote block-level storage resources. This demonstration enhances storage efficiency and exemplifies how Android devices can evolve into versatile hubs capable of controlling IoT devices and seamlessly centralizing data. Such experiments lay the

foundation for more efficient and adaptable Android ecosystems, catering to users' ever-evolving storage and data management needs in today's digital landscape.

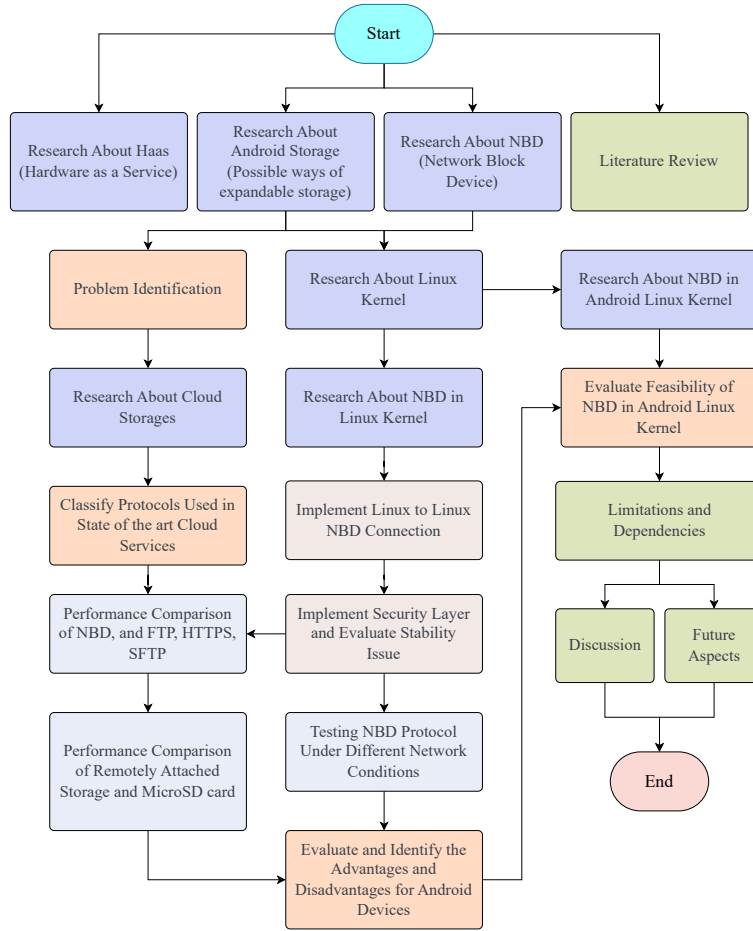


Figure 1.2: Workflow Diagram of NBD for Integrating on Android

1.4 Research Structure

We divide our paper into several sections to enhance its organization. Section 1 contains the background, motivation and research contributions to integrate remote server storage on the Android operating system using NBD. Following that, in Section 2, we explain the terminologies such as *nix philosophy, Linux Kernel, LKM, Android Kernel, NBD, SELinux, Symlink and Waydroid. Section 3 contains similar concepts and comparisons based on findings. Section 4 proposes our methodology and security architecture. Section 5 shows the implementation according to experimental evaluation needs. Section 6 elaborates on the results by testing file transfer performance and a game. Furthermore, we discuss the possibilities and challenges for remote Android storage servers in Section 7. Finally, we state the future prospects, impacts of remote storage servers using NBD and the conclusion of our research in Section 8.

Chapter 2

Context & Overview

2.1 *nix Philosophy

*nix is a lineage of UNIX-based operating systems. It behaves and follows the manners of UNIX. UNIX was introduced in 1969 with a philosophy of "Do one thing, do it better" [32]. Following that philosophy, UNIX has introduced a simpler mechanism for interacting with hardware peripherals through file I/O operation. *nix has also shown how simple virtualization can be and how the file system can be implemented based on that. *nix, like the operating system, has mainly the following types of file system implementation, which are [9]:

1. Special Files
2. Ordinary Files
3. Sockets
4. Directories
5. Symbolic Links
6. Pipes

The "Special Files" implementation of those three types of file systems has the most distinctive features and we are interested in it. All the UNIX systems support I/O operation with one of those systems and normal disk files require permission to read and write [2]. This basic feature is available in "Special Files" but the core unique part is any requests to read or write special files cause the corresponding device to be activated. There is an entry directory using special files where all the files reside under /dev directory [2].

2.1.1 File System

In the Unix operating system, regular files, directories, devices and network sockets are represented and accessed through file abstraction. Consistency facilitates a straightforward method of engaging with diverse data types. The design of Unix file systems is characterized by modularity and simplicity, where each component fulfills

a distinct function. The modular architecture facilitates the system's seamless management, expandability and compatibility. It enables users to adapt and provide options for end-users. We can choose and adjust various modules according to their requirements, establishing a personalized system that meets their demands. The flexibility also applies to the development process because multiple teams or people can concurrently work on various modules, fostering collaboration and enabling parallel development.

2.1.2 Symbolic Links

The Unix system also supports the implementation of the mounting concept. It means diverse file systems can be connected to designated directories within the hierarchical structure. The idea of abstraction allows seamless access to storage devices and external file systems, leading us to Symbolic Links. The feature referred to as soft links, sometimes known as symbolic links, is present in Unix-like operating systems. It enables the creation of a specific type of file that serves as a pointer to another file or directory. Symbolic links efficiently access or traverse the file system as shortcuts or references to the desired file or directory. When users access or open a symbolic link, the operating system seamlessly sends them to the designated target file or directory. Symbolic connections can extend across disparate file systems and even reference non-existent files or directories. In Unix systems, it is widely employed for diverse functionalities such as establishing expedient links to frequently accessed files, arranging file system structures and facilitating flexible file location management. In our research, we have used the symbolic link concept to access the files of remotely located NBD (Network Block Device) storage files.

2.1.3 User Space and Kernel Space in Linux

Another important *nix philosophy crucial to our research is the user space and kernel space separation. The user space is the defined environment for running user applications and the kernel space is defined for containing the operating system kernel. The process of separation serves numerous essential purposes. First of all, it offers memory protection. The kernel works in privileged mode, granting it direct access to system resources. Conversely, user programs operate in a restricted mode, which limits their access to system resources. This measure guarantees that user programs cannot disrupt or compromise essential system resources. Additionally, the act of separation facilitates hardware protection. The kernel directly controls hardware devices and oversees access and utilization. In contrast, user programs establish an indirect interaction with hardware through the kernel, which serves as a mediator and ensures appropriate access and utilization. Moreover, the clear distinction between user and kernel space facilitates effective task management. The kernel manages low-level activities and offers various resources and services to user applications. User applications can initiate system calls to request specific services and it is the responsibility of the kernel to execute the required activities on behalf of the applications. Our research opens the door to reanalyzing using the NBD module in the kernel space for a unique purpose to amplify the available options for cloud storage serving the Android community.

2.2 Linux Kernel

The Linux kernel's file system exhibits a modular and extensible design, enabling the incorporation of novel file system kinds and functionality. The inherent flexibility of Linux allows for seamless compatibility with a wide range of storage devices. It contains traditional HDD (hard drives), modern SSDs (solid-state drives) and NAS (network-attached storage) systems. The storage devices are locally denoted as files under the `/dev` directory. These files are sometimes referred to as device files or device nodes. Every device file is associated with a particular physical device or device driver. Device files are generated either during the initialization of the system or upon establishing a connection with a new appliance. Interfaces are designed to facilitate the interaction between the operating system and users, enabling them to engage with various devices. The device files provide both read and write operations on the corresponding devices. In the Linux operating system, device files are classified into two distinct categories: character devices and block devices.

2.2.1 Block Devices

Block devices are put to use to facilitate random access to data. They are usually used to attach solid-state drives (SSD), hard drives and sometimes USB drives. Fixed-size blocks are permitted for both the reading and writing of data. Block devices offer a file system interface that enables efficient access and management of data stored on these devices by the operating system and applications. Linux encompasses various block devices, such as `/dev/sda`, which represents a hard drive and `/dev/nvme0n1`, which denotes an NVMe SSD. The device files are in the `/dev` directory and function as intermediaries between the user and kernel spaces. It enhances the standardized access and management of the devices.

2.2.2 Character Device

Character devices are utilized to access data sequentially and serve as interfaces for devices that transmit data as one-by-one characters. Examples of such devices are terminals, serial ports and printers. In contrast to block devices, character devices lack a predetermined block size and do not provide the capability for random access. In contrast, these entities offer a continuous sequence of characters that can be read or written. Character devices are frequently employed in real-time or streaming data processing. Linux encompasses various character devices that are essential for interacting with the system. Two prominent examples of such character devices are `/dev/tty`, which facilitates communication with a terminal and `/dev/ttyS0`, which enables communication through a serial port.

With the help of the Network Block Device philosophy, NBD enables network-based storage solutions by exporting block devices from a server to client systems. The client systems can treat the remote storage in a similar way to devices that are connected locally. The NBD protocol operates at the block level, transferring data in fixed-size blocks rather than individual characters. This makes it suitable for accessing remote storage devices efficiently.

2.3 Loadable Kernel Modules (LKMs)

Loadable kernel modules (LKMs) serve as a means to incorporate supplementary functionality into the kernel, thereby establishing a distinct barrier for compartmentalization. We do not notice any significant difference at the source level when comparing the core kernel code and the Loadable Kernel Module (LKM) code. So, both can be compiled in a general way into the main kernel image. On the other hand, the bootloader loads the primary kernel image and the Loadable Kernel Modules (LKMs) separately, so the LKMs do not form any part of the core kernel image. In contrast, the kernel exhibits the behavior of dynamically loading a Loadable Kernel Module (LKM) only when it necessitates the particular capability provided by said LKM. Hence, throughout the execution of a program, a distinct and coherent distinction exists between Loadable Kernel Modules (LKMs) and the remaining components of the kernel. However, the monolithic architecture of the kernel effectively eliminates this separation [31]. The next stage is of particular significance importance to the loading mechanism.

The current task is preparing the module's symbol table to provide access to potential future modules. It is necessary to acquire a comprehensive understanding of the precise data structures. As shown in 2.1, the definition of each particular module is characterized by a module structure. Several primary fields are introduced. The fields include:

- Name of the module
- Next pointer module in the linked list
- Size of the module
- Number of dependents
- Number of symbols

A pair of structures should be mentioned as “`module_symbol{}`”, which is utilized to locate the exported symbol table of the module and “`module_ref{}`”, which is important for maintaining dependence information [1].

Every Loadable Kernel Module (LKM) is kept as an individual file within the file system. The files are commonly denoted as “.ko” file extensions. The files include the compiled code and data necessary for the module. A module will load when a user or system process uses the `insmod` or `modprobe` commands to call on the kernel's module loader. The loader is responsible for locating a specified module file and, upon successful discovery, loading it into the kernel's address space. Once the module has been loaded, it can be initialized by invoking its initialization function. The function mentioned above commonly registers the module with the kernel, establishes necessary data structures or resources and readies the module for utilization. The loaded module can incorporate additional kernel system calls, device drivers, or other functionalities. As a demonstration, a module could enhance the functionality of a system by integrating compatibility for a novel network card, file system, or hardware sensor. Lastly, the module's functionality becomes accessible to apps in user space. Applications can engage with the module using system calls or other mechanisms made available by the module.

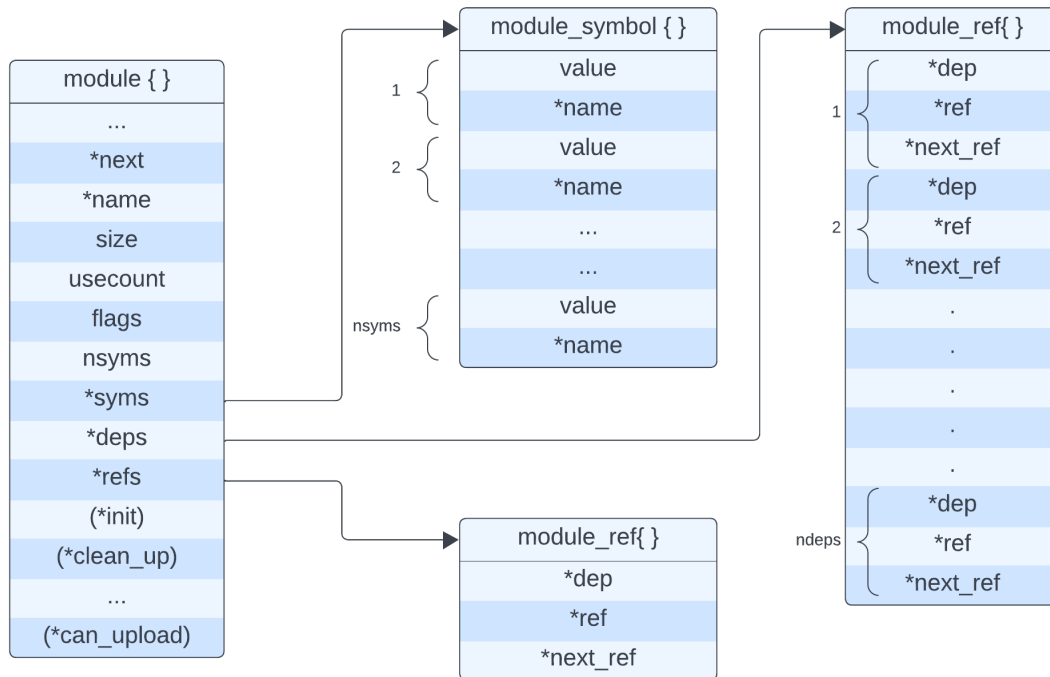


Figure 2.1: Data structures in kernel development for module implementation [1]

2.4 Android Kernel

Android is built on top of the Linux kernel. That said, the Android kernel is very similar to the Linux kernel in architecture and functionality. The Unix operating system is the foundation for both kernels, sharing similarities in many ways. The Android kernel is a customized Linux kernel tailored for portable electronics. Process management, memory management and device drivers are just a few similarities between it and the Linux kernel's features and functionalities. Mobile-specific features like power management and support for touchscreens and other input devices are also included.

One of the key similarities between the Android kernel and the Linux kernel is their open-source nature. Both kernels are released under open-source licenses. So, the source code is available in public privacy, where anyone can modify, view, or distribute it. This has caused a large group of developers and contributors to work to improve and enhance the functionality of both kernels. Overall, the Android kernel is very similar to the Linux kernel in architecture and functionality. However, it includes additional features and optimizations specific to mobile devices.

The Android platform has been further developed with many libraries in addition to the Linux kernel for enabling complex functionality; see figure 2.2. These libraries were sourced from numerous open-source projects. The Android team created their own C library to deal with licensing issues. Moreover, the "Dalvik Virtual Machine" is a Java runtime engine they developed that has been optimized for mobile platforms with limited resources. Finally, the end-users were provided with an application framework that can give them access to system libraries.

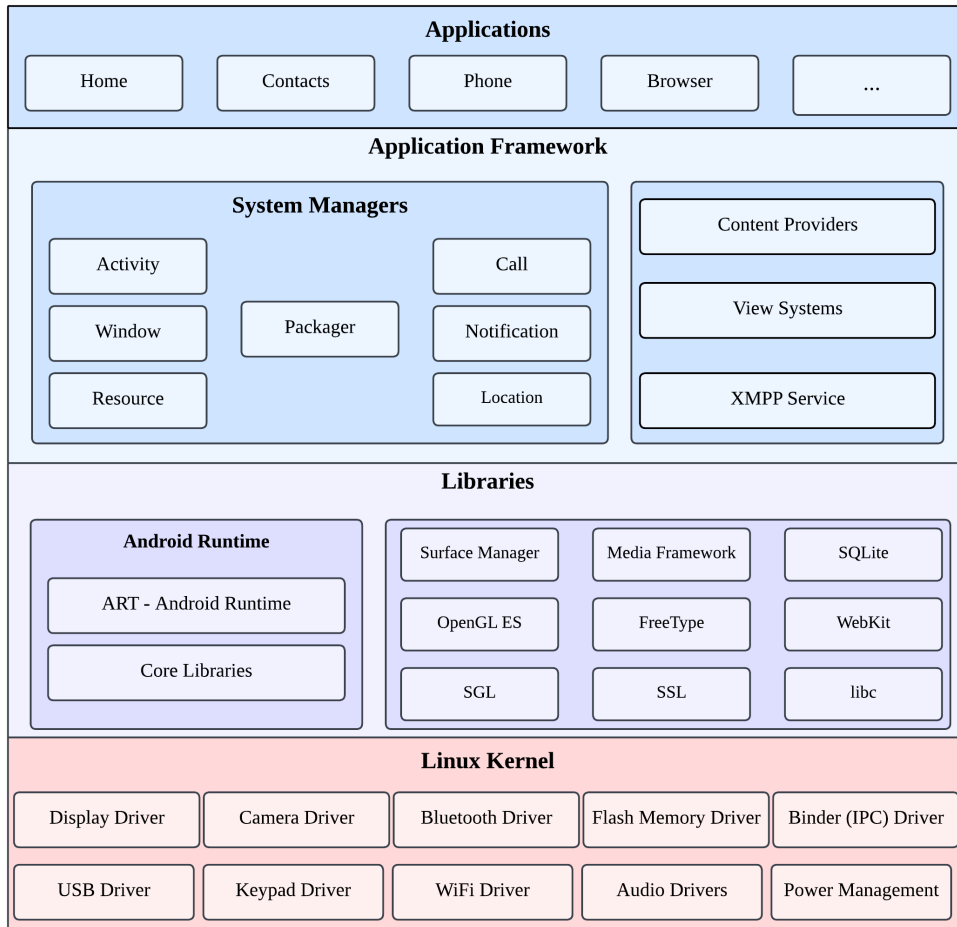


Figure 2.2: Android Architecture, Linux Kernel & Libraries [12]

2.5 Network Block Device (NBD)

The NBD suggests an access model, which can be seen in the figure 2.3 that replicates a block device, specifically a hard disc drive (HDD) or a hard disc partition, on the user's local machine. However, it connects to a geographically located remote server, which provides the actual physical storage over the network [3]. This model can be seen in the NBD.

The NBD protocol operates by breaking the block device into manageable "blocks" and sending them over the network as needed. The remote machine can then access these blocks as a local block device. As a result, the network's bandwidth can be used more effectively because it is only used to transmit necessary blocks.

NBD is widely utilized in virtualization environments because it enables access to block devices hosted on distant machines by virtual machines. Additionally, it can be used in distributed storage systems, enabling numerous machines to connect to a single shared block device over a network. As there is an involvement of networks, the question of security arises inevitably. Security for Network Block Devices (NBD) in Linux systems is primarily managed through a combination of access control mechanisms and encryption; these can be:

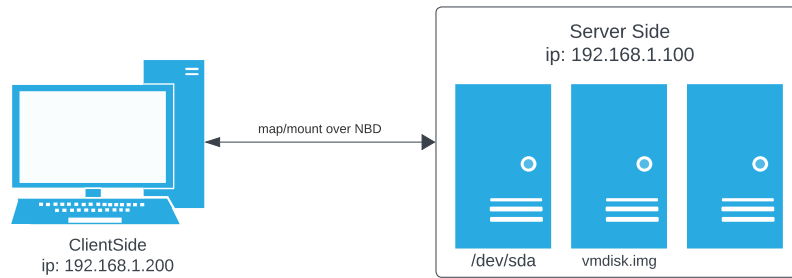


Figure 2.3: Network Block Device

2.5.1 Access Control

NBD (Network Block Device) devices are managed by adhering to the established Linux file permissions [30]. This implies that the NBD devices can be accessed and managed at the file level, similar to any other file within the system [3]. File ownership and permission settings can be employed to impose limitations on the accessibility of NBD device files for specific groups and users.

2.5.2 Network Security

Network security is paramount in Network Block Devices (NBD) as it runs across a network [30]. Utilizing established network security protocols and techniques can result in network-level security. Frequently, this entails implementing firewalls, using Virtual Private Networks (VPNs) and adopting additional network security protocols [3].

2.5.3 Authentication

In particular, Network Block Device (NBD) configurations authentication procedures are sometimes employed to guarantee that only clients with proper authorization can access NBD devices [3]. This may encompass using login and password authentication and alternative authentication methods, such as SSH keys.

2.5.4 Encryption

Encryption can address data privacy and confidentiality concerns in the context of NBD traffic. One potential method for achieving this objective involves the utilization of SSH tunneling as a means to encrypt Network Block Devices (NBD) communication. The utilization of SSH for tunneling NBD enables the secure transmission of encrypted data across the network. However, implementing encryption decryption functionality at the kernel level may not be feasible because of the minimal memory allocation at the lower level kernel.

2.5.5 NBD Server Configuration

The proper configuration of the NBD server is of utmost importance in ensuring security. The process encompasses establishing access controls, determining permissible hosts or IP addresses that can establish a connection with the NBD server and configuring authentication processes.

One can set up Oracle Real Application Clusters using only standard PC hardware and Linux operating systems via NBD [37]. In Linux NBD, we only have access to a single client at a time to access the identical block device [3]. Furthermore, the NBD shares common branches such as Extended-Network-Block-Device (ENBD), then it shares with Global-Network-Block-Device (GNBD), lastly shares with Distributed-Replicated-Block-Device (DRBD) etc [26]. They are responsible for different use cases of NBD depending on the demands of the cloud architecture [30]. The adaptability of NBD is one of its main advantages. It can be used with various block devices and network setups and is simple to incorporate into current systems. NBD is also an open protocol, meaning anyone can use it and is not dependent on any vendor or platform.

2.6 NBD as Loadable Kernel Module (LKM)

The Network Block Device (NBD) is a module inside the Linux kernel that facilitates the remote access of block devices over the network [3]. Although NBD is frequently utilized in Linux operating systems, it is not typically implemented as a Loadable Kernel Module (LKM) for Android kernels. The Android kernels are purposefully customized for mobile devices, exhibiting distinct requirements and configurations compared to conventional Linux kernels. The Android kernel has been designed to cater to the Android operating system's unique hardware and software needs. Integrating NBD support as an inherent feature in the Android kernel is feasible instead of utilizing it as a loadable module. However, this would require modifying the kernel source code and rebuilding the kernel specifically for the Android device you are working with.

Network Block Device (NBD) is not natively supported in the Android kernel and there are no official indications of it being deprecated or planned for inclusion. The Android kernel primarily focuses on the Android operating system's needs and hardware requirements. NBD, which is typically used for networked block storage. As there are devices with limited local storage, NBD could obsolete the limitations imposed by the bounded storage as it provides a means to access data stored on remote servers, cloud storage, or network-attached storage [1].

2.7 Android SELinux Policy

The Android SE (Security-Enhanced) Linux Policy is a security framework implemented on the Android operating system based on SELinux (Security-Enhanced Linux) [15]. SE Android augments the security of the Android platform by implementing mandatory access controls and rules. These measures effectively restrict and regulate the behavior of processes and applications operating on an Android device.

SE Android, which stands for Security-Enhanced Android. It is a robust security framework designed to fortify the Android operating system against various security threats. The system primarily functions based on the Mandatory Access Control (MAC) notion, which represents a deviation from the traditional Discretionary Access Control (DAC) approach, in contrast to discretionary access control (DAC), which grants resource access decisions primarily to the owner [15]. MAC imposes a central control mechanism based on predefined and stringent policies.

One of the critical uses of this technology is isolation and sandboxing. SE Android is proficient in establishing unique security contexts for each Android application and system function. Each application operates within its distinct sand-boxed environment, imposing substantial limitations on its ability to access fundamental system resources, as well as the data and functionalities of other applications. Implementing isolation measures enhances overall security by mitigating the possible harm a hacked application can cause.

Furthermore, SE Android enforces privilege separation, which is crucial for minimizing the attack surface and reducing the impact of potential security vulnerabilities. This separation involves restricting certain system components and apps to only the minimum privileges they need to operate effectively. By adhering to the principle of least privilege, SE Android significantly mitigates the risk associated with privilege escalation attacks and other security breaches.

One characteristic of SE Android is its capacity for policy customization. Manufacturers and system administrators can customize SE Android's security policies to align with the unique security needs of a device or system. The ability to adapt facilitates the precise adjustment of security restrictions, ensuring their alignment with the Android device's specific usage and threat environment.

Moreover, SE Android provides a framework for implementing security enhancements to the Android platform. These enhancements encompass a wide range of features and mechanisms, one of the most notable being system call filtering. By controlling system calls, SE Android can proactively prevent certain actions that might otherwise lead to security breaches, further bolstering the platform's overall security posture.

SE Android is a notable advancement in bolstering the security of Android devices. SE Android enhances the protection of the Android operating system by implementing measures such as MAC (Mandatory Access Control), isolation, privilege management, policy customization and a range of security-oriented features. These measures strengthen the system's ability to withstand diverse security threats and attacks.

2.8 Our Adhered SELinux Policies

Regarding utilizing the Network Block Device (NBD) as a Loadable Kernel Module (LKM) for remote storage on the Android platform, we have adhered to a set of considerations and policies:

- **Access Control:** We have followed Android's access control mechanisms and SE Android policies when loading the NBD module. Ensured that only au-

thorized users or processes could load and use the module by implementing and enforcing Security Policy Language (SPL) [15].

- **Module Signature:** A potential method exists for digitally signing the NBD module to guarantee its integrity and authenticity. The Secure Boot techniques employed by the Android operating system may necessitate loading modules that have been signed.
- **Testing and Validation:** These were conducted on the NBD module within our prototypical Android device and kernel to ascertain its stability and compatibility. In the absence of adequate testing, the introduction of custom kernel modules may result in the manifestation of instability.
- **Documentation:** We have comprehensively documented the loading process and the NBD module to facilitate future reference and address potential troubleshooting scenarios. This documentation encompasses several aspects, such as security considerations and access control policies, which are crucial for ensuring the integrity and confidentiality of the system.
- **Device Rooting:** Depending on the device and Android version, a user may need to root the device to load custom kernel modules. But one must understand the implications and risks of rooting before proceeding.

2.9 Symbolic Linking Philosophy

Symbolic links are a feature that improves the transparency and flexibility of file and directory management in Unix-like systems. The system permits users and administrators to conveniently generate references to files and directories, providing an extra degree of abstraction. The Unix philosophy places significant emphasis on the principle of modularity, which entails that each component should excel at executing a singular task. Symbolic links align harmoniously with this philosophical approach by offering a means to modularize and disentangle file structures. Using file or directory replication facilitates the existence of a single instance of data in several locations, enhancing operational efficiency and reducing complexity. The single program tree, a tree of symbolic links, namely `/System/Index`, assembles the instances of the files from each program in the system. [29].

Two types of linking are introduced in the Unix philosophy: soft links (symbolic links) and hard links. Hard links are a file system feature wherein various directory entries, also known as filenames, are related to an isolated inode. An inode is a data structure representing a disk file [5]. In essence, there are distinct file designations denoting the same data. Soft links are references to files or directories by name. They act as shortcuts or pointers to the target file or directory. Symbolic connections have the advantage of being able to span several devices and file systems, unlike hard links, which are limited to the same file system. Those above characteristics endow them with the ability to serve as adaptable instruments for referencing across various devices. Relevant and significant symlinks or network device labels deliver a course to identify devices based on their configured properties or latest configuration [39].

2.10 Waydroid

Waydroid is a free and open-source project that aims to run Android apps on Linux distributions [40]. It provides a compatibility layer between Android and Linux, allowing Android apps to be executed on a Linux system. As Waydroid focuses on app compatibility rather than kernel-level modifications, our goal is to enable the NBD module support in the Android kernel to allow the Android system to attach block storage devices remotely. Using the available support for Waydroid, we have established a bridge between Waydroid's Android environment and the host Linux system. This enables seamless data sharing between the Android and Linux environments, allowing Android apps to interact with Linux files and vice versa. It opened the door to remote data access for Android devices without using any third-party applications in the user space. Users can also migrate Android app data from one Waydroid instance to another or from one host system to another; NBD can simplify the process by allowing you to transfer data over the network. But the most important and noticeable attribute we have achieved is the expandable cloud storage connected at the kernel level.

We can mirror Android systems running on remote servers and access them as local devices if we implement Waydroid on top of Linux. This can be useful for scenarios where you must maintain synchronized copies of Android systems across different locations. We can also perform operations such as resizing partitions, creating file systems and managing block devices on remote Android systems. By combining Waydroid with the NBD module, you can create virtualized environments where Android systems run on top of a Linux host. This can benefit testing, developing and running Android apps on Linux.

Chapter 3

Related Works

3.1 Background

Android devices offer a range of storage solutions to cater to different user needs. Let's discuss the existing storage options available on Android devices:

1. **Internal Storage:** Android devices typically come with built-in internal storage, which is non-removable and used to store the operating system, system apps, user data and installed applications. Internal storage provides fast access to data and is usually available in different capacities, ranging from a few gigabytes to several hundred gigabytes.
2. **External SD Card:** Many Android devices support expandable storage through external SD (Secure Digital) cards. Users can insert an SD card into a designated slot on the device to increase storage capacity. SD cards come in various sizes and speeds and can store media files, documents and other user data. However, not all Android devices support external SD card storage, as some newer models may omit this feature.
3. **USB On-The-Go (OTG):** Android devices with USB OTG support can connect to external USB storage devices such as USB flash drives, external hard drives, or SD card readers using a USB OTG cable. This allows users to access and transfer data between the Android device and the connected external storage.
4. **Cloud Storage Services:** Android devices provide seamless integration with various cloud storage services such as Google Drive, Dropbox, OneDrive and others. Users can upload their files and data to the cloud, allowing convenient access from multiple devices and providing an additional backup solution. Cloud storage often comes with a limited amount of free storage, with options to purchase additional storage capacity if needed.
5. **Network Attached Storage (NAS):** Android devices can access network-attached storage devices on the local network. By connecting to a NAS, users can access shared folders and files on the network, enabling centralized storage and file-sharing capabilities.
6. **Adoptable Storage:** Introduced in Android 6.0 Marshmallow, the adoptable

storage feature allows users to merge the device's internal storage with external storage (like an SD card) to create a single unified storage pool. This feature treats the external storage as an extension of the internal storage, increasing the overall available space.

It's important to note that storage options' availability and specific features can vary across Android device models and manufacturers. The Android version and device specifications may also impact the supported storage options.

In the modern era of technologies, hardware components are getting up to date incessantly. For example, we get new components almost yearly to utilize the current market demand. Multiple studies have been done regarding distributed component sharing or sharing files over the internet. Moreover, as researchers always try to find an efficient solution continuously for upcoming complex problems, after studying multiple types of research and their limitations, we are proposing the HCaaS model. Currently, everyone is fascinated by cloud computing. If we move back to our past and find the limitations of that research, we can overcome those complex problems with the latest technology. Furthermore, it is nearly impossible to demolish all the challenges. However, we are only focused on improvising the sharing system of the Hardware Component as a Service (HCaaS).

3.2 Taxonomy of NBD, Linux and Android

This section presents a taxonomy of NBD, Linux and Android studies. It encompasses various aspects such as storage expansion by the method based on Network File System, storage expansion by the method based on File Transfer Protocol, Storage expansion using Flash Storage, the usability of cloud storage on the server side, the usability of cloud storage depending on users in different platforms, cloud storage using SaaS, storage cache hierarchies in shared server farms, virtual machine migration via WAN, Linux based implementation, Linux vs. Android, x86 vs. ARM, build-ability of NBD with higher performance, network storage protocols for reduced protocol overhead, security and data privacy, I/O parallelism, user space and kernel space, distributed source sharing, high-performance distributed systems and loadable kernel module. These studies contribute to understanding NBD, Linux and Android characteristics in different aspects, enabling better decision-making.

3.2.1 Android Storage System Expansion

In this section, we talk about different methods of expanding the storage systems of Android, like network file systems, file transfer protocols and flash storage.

Storage Expansion by Method Based on Network File System: The study [17] has proposed a solution for storage systems of mobile devices by implementing RFS, a network file system optimized for wireless communication, on Android and the cloud. The RFS file system that the authors have proposed is a client-centric design that allows mobile devices to store files in the cloud. RFS uses TCP/HTTP protocol, while NFS and Coda use TCP/UDP and UDP, respectively, allowing RFS to transmit more packets than the others. The performance of RFS is tested with Wired, WiFi and WCDMA networks over RFS, Coda and FScache,

where RFS performs better than the other two. The RFS is evaluated on an Asus EeePC 1000H netbook that utilizes 1GB of memory and a 1.6GHz Atom processor and is measured on Ubuntu Mobile. The privacy overhead is also evaluated over WiFi, WCDMA and Wired networks, while Android booting is done over wired, native, WCDMA and WiFi.

Storage Expansion by Method Based on File Transfer Protocol: This study [18] proposes FTP4Android, an FTP client aimed to solve the limited storage issue in Androids. With this system, users can upload, download, delete and create folders for their data. Additionally, they developed a treasure hunt application to demonstrate the usability of FTP4Android. With this, a user (the master) can use the application in the remote file system to store a file (such as a clue, a treasure map or a reward). To begin with, the file system root folder is already set up in the application to be inside a localized folder that permits writing files and folders. Subsequently, it is set up to use a specific set of remote FTP servers. However, there is only one restriction: the FTP servers that are selected must support two connections that are simultaneous at the least. Nonetheless, to perform operations on each FTP server in the list of servers, a folder or directory is created to prevent an inconsistent remote file system. Uploading files generates chunks of the file. However, downloading files would create a download thread for each chunk. For deleting, a thread is created to remove the specified chunks. Ultimately, using threads is feasible to utilize all the uploading and downloading bandwidth fully. The cons of FTP4Android depend on internet connectivity, the unavailability of a utilized free server and inconsistency of the visualization in the file system. Furthermore, the time it takes to download, upload or remove a file from the distant file system is tested under two scenarios. One of them is when the quantity of fragments the file is chopped into is varied. The other is when the quantity of threads in action is varied.

Storage Expansion Using Flash Storage: The study [19] demonstrates that several widely used applications, including Facebook, email, maps, Web surfing and app installation, are affected by storage performance by implementing and thoroughly analyzing the I/O characteristics of device program on flash storage systems and Android-based handset to tackle the storage performance issues. To conduct this study, the authors created a measuring infrastructure for Android and modified the Linux kernel to provide resource utilization data. Using MonkeyRunner, they create a benchmark harness for automated testing of GUI-based apps. According to their comparisons, SD cards' "speed class" designation is not always a reliable indicator of application performance. They also noticed higher overall CPU usage for the identical application when using less speedy cards. The cause might be traced to weaknesses in the network, storage subsystem, or both. Next, they found that efficiency is achieved by imposing a modest quantity of knowledge in terms of domain or application, for instance, in the occasion of test solutions. The intended device should be attached to the host computer over TCP/IP or USB in the debugging mode. Eight detachable cards with microSDHC, two from each of the four SD speed groups and an internal, permanent flash storage were used for the experiment. They performed tests with and without the measuring environment to determine the overheads of the apparatus and discovered that the modifications only add less than 2% of runtime overhead. Program runtime performance, concurrent program, pro-

gram launching and CPU metric consumption are used to conduct the experiments. A glaring example emerged through what-if analysis: SQLite’s overbearing control of program caches. Writing a cache map in the web browser to SQLite slows down the writing of the caches significantly.

3.2.2 Storage I/O in Cloud Environment

This section discusses cloud storage and its usability on the server side and through SaaS. We also talk about storage cache hierarchies and virtual machines.

Usability of Cloud Storage on the Server Side: In this paper [17] different usabilityes of cloud storage are illustrated; for example, the files or RFS images that reside in the mobile devices include user files (/usr/file) and system files (/bin/app) which can be in the cloud storages. The RFS client interacts with the RFS server that retrieves resources into a local cache to store the user information in the cloud [17]. Unlike the close-to-open stability of MFS and LBFS, the suggested program RFS inconsistently caches data until the user requests an explicit synchronization. In the server model of RFS, features like cloud cache and adapter are implemented to better map the performance characteristics in mobile device usage. The cloud cache is a remote cache that amplifies the performance of accessing the file made by mobile device users and the cloud adapter is a remote cloud adapter that hides the cloud storage system interface variety [17]. On the other hand, the server of RFS is directly set up on the cloud storage.

The following study paper [16] describes a mechanism for distributing resources under the I/O borderline. This technology can be used on platforms that are very different from each other. They have prioritized using a personal working environment with I/O peripherals so that users can use smartphones and PCs employing different interconnections to make a fully personal Cloud. They have made the initial hardware resolution for smooth pro-migration PCIe I/O access for peripherals, showing that the idea works and can make a big difference for the end user. They have also examined the logic that can be combined for personal Cloud Experiences.

The following paper [14] talks about the vast amount of information/data provided through cloud computing. This study discusses building a specialized resource-sharing technique that allows cloud members to share I/O devices by designing composable USB. A transferred application operating on the targeted host can access the USB I/O devices on the source host, as the USB was created with a special design. Moving VMs to various physical computers may change the pool of resources thanks to the live VM migration functionality. This paper has also mentioned virtualization technologies such as VMware Vsphere, Xen and KVM. They have discussed one problem with sharing I/O peripherals: the necessary computing and I/O resources might be on different physical machines. This is a case of resource sharing that a basic migration can’t handle. A USB that can be folded and used with personal cloud apps was made and tested to solve this issue. When a program requires computing assets and I/O peripherals at several physical systems, pro-migration IO sharing makes it easier for the cloud to arrange and assign resources.

Platform-Based Cloud Storage Usability: This study [27] investigates the efficiency of data transfer and the system’s objects by looking at a dataset of logs containing 349 million HTTP requests made by 1.1 million individual mobile

device users within a wide-ranging storage service over cloud. Their analysis includes sessions and burstiness where individual users' inter-file operation times are modeled by a Gaussian mixture with two components, one corresponding to in-session intervals and the other to intersession intervals. It also includes storage-dominated access behavior, showing that mobile users seldom conduct both operations in a single session. Instead, they solely store data files that occurred in 68.2%

Cloud Storage Using SaaS: This paper [34] proposes a system that helps users to share device-oriented information and data with the help of a private cloud and SaaS. This system primarily focuses on delivering information between different devices where a website is available for the users to access their account and can retrieve data such as missed call history, texts, battery status and GPS information. An Android application is also integrated into the system to update the cloud and web requests.

Storage Cache Hierarchies in Shared Server: In this research [10], a novel method for managing application interference in shared server farms' cache hierarchies is presented. The authors designed and implemented a mechanism for partitioning storage caches and the buffer pool dynamically online. To begin with, a cache controller planted in the DBMS activates the segmentation of the two caches. In addition, they wanted to assess the relationship involving cache allocation limit settings and the resulting Quality of Service (QoS) for each program dynamically. According to the authors, this paper aims to locate a configuration that enhances the overall usefulness of a certain group of apps linked to the Service Level Objectives (SLOs). Meanwhile, the RUBiS online bidding benchmark and the TPC-W e-commerce benchmark are two applications that the authors of this paper utilize in their tests. They employed these two applications and scheduled them to employ a unified DBMS sample and the storage server. They also test the MySQL database engine. Moreover, they contrasted two approaches: SHARED and IDEAL. In SHARED, programs distribute the DBMS, the storage cache and the buffer pool without quota implementation. On the other hand, in IDEAL, they experimentally iterated through all feasible partitioning configurations of both caches. Through this, they selected the best setting where they met the SLO for TPC-W. Besides, approximative performance models called application surfaces were constructed. One reason is that mapping cache allocation limits the delay experienced by the program and its corresponding utility for every program. It is also done as part of an effective self-executing hunt for the ideal cache with a tiers partitioning solution. Another reason is that each program's performance models are leveraged to answer "what-if" cache segmenting instances for every app collection. Moreover, MySQL utilizes Standard Linux system operations and drivers such as iSCSI or NBD (network block device) to interconnect with the virtual storage device. In this study, the authors alter the initial NBD processing module on the remote server, which is utilized in Linux for virtual disk access, to alter the NBD data package into their internal protocol packets.

This paper [13] works with an efficient multi-resource assignment method built on a unified resource-to-performance design that takes into account pre-existing general system knowledge and resource dependencies, such as those caused by cache replacement policies, the tracking of application and recording of baseline system are documented on the internet. The authors demonstrated through experiments utiliz-

ing a variety of common e-commerce benchmarks and simulated workloads that their performance model is accurate enough to converge to a nearly ideal global partitioning solution quickly. The following disk schedulers' performance isolation guarantees have been built and compared: Start-time Fair Queuing (SFQ), Quanta-based scheduling, Lottery-based, Earliest Deadline First (EDF) and Façade. The authors continuously run two simulated workloads on the storage server: a small workload (Workload-A) with one outstanding request and a big workload (Workload-B) with ten outstanding requests. Without any notable changes to the DBMS and no specific alterations to the existing interfaces between each component, the method used in the study automatically establishes data quotas for each application in the database and storage caches in a clear way. To achieve this, the authors track all I/O operations at the extent of the DBMS buffer pool and periodically sample the average disk latency of each application in a baseline configuration in which the application is given complete access to the disk bandwidth. The authors used a single-level cache model to simulate the cache hierarchy. They specialized this model for the two most widely used or suggested cache replacement policies: uncoordinated LRU and coordinated Demote. The authors implemented access trace collecting and computed the miss ratio curve (MRC) only at the buffer pool level to derive a comprehensive resource-to-performance model. The virtual storage system prototype that makes up the infrastructure (Akash1) is made to function on affordable hardware. Any storage client, including file systems and database servers, can access data from numerous virtual volumes with its support. It reads and writes logical blocks from the virtual storage system by employing the Network Block Device (NBD) driver already implemented in Linux [13].

Virtual Machine Migration via WAN: This paper [11] proposes a cutting-edge storage access technique that significantly encourages live VM migration via WAN. According to the authors, it frequently oscillates between VM disks between the source and the destination having the minimum influence on Input/Output performance. To begin with, the main problem with wide-area live migration is the consistency of Input/Output operations in the virtual disks before or after migration. The suggested mechanism addresses this. So, the suggested mechanism functions as a server dedicated to data storage for a block-level storage Input/Output protocol (like iSCSI and NBD). The implementation prototype is known as xNBD. In addition, the Linux kernel's netem module was implemented between two distinct networks to simulate a WAN scenario. To conduct this study, the configuration parameters were 120ms RTT and 100Mbps bandwidth. Moreover, a virtual machine host node is connected to its storage server through a 1Gbps Ethernet connection in the source and the destination's local networks. Furthermore, after launching a virtual machine (VM) that has 512 MB of memory and a 4GB virtual disk that is formatted by ext3 from the source, a Linux kernel compile was started. Initially, they began migrating the VM to the target site at a specific time. Then, the WAN bandwidth was used for memory transfer traffic for about 80 seconds. In the end, the VM at the destination site was restarted once memory migration was finished. In conclusion, this paper is similar to our work utilizing Linux kernel and NBD protocol. However, our work delves deeper into these subjects.

The following journal [25], the authors set up a multi-GPU-based management framework that dynamically allocates cloud resources based on their needs. The

authors made a GPU load balancing algorithm that uses DMLS-GPU (Dynamic Multi Load Status) to determine how much work each GPU is doing and how much it is being used. The architecture has three layers: the User layer, the Computing Resources Service layer (CRS) and the Virtual Resource Management (VRM) layer. Comparing physical GPU and virtualized GPU, the paper analyzed the performance of virtualization, an evaluation of the ability to scale and an analysis of the system's performance. To sum up, the DMLS-GPU algorithm is suggested and AHP is used to examine its parts. Soon, the corresponding change will be made to make the system run more smoothly.

This research [4] demonstrates in detail how an operational computer, including the state of its CPU, RAM, disks, registers and I/O devices, can be rapidly transferred over a network using the metaphor of a capsule. Even though x86 capsules may hold terabytes of disk data and hundreds of megabytes of RAM, x86 computer systems have been chosen for transfer because they are common, inexpensive, can run the apps used in this research and provide migration tools. A prototype system has been used to illustrate these optimizations, which construct and execute x86 capsules employing VMware GSX Server's VM monitor that works with networks with a 384 kbps maximum speed. A unified program uses serialization and the capsules' flexibility to deliver client backup, hardware management and software administration. This study demonstrated a technique that transfers the state of a computer across a sluggish DSL link in minutes instead of hours has been demonstrated in this study and the experiments show that the capsules move over a 384 kbps DSL link in a maximum of 20 minutes or less.

This study [8] demonstrates effective ways to create virtualization of the I/O subsystems and peripheral devices. The authors of this study suggested a highly efficient solution for I/O virtualization and the self-virtualized system method that is intended for I/O virtualization to have higher performance. With the help of the system's internal processing data, the consistency in implementing SV-NIC ensured great scalability and overall performance. It also allows flexible and effective mappings of virtualization features by adding self-virtualized I/O to the definition of a self-virtualized device. The paper talks about The CDNA prototype and The OSA network interface as examples of similar work that could be done. This study looked at ways to make the system even better and more organized, such as substituting micro-engine programmed Input/Output by employing DMA to make the upstream VIF work better. This paper also improved TCP efficiency by employing TCP segment offload and deploying more support for large Maximum Transmission Unit (MTU) sizes.

3.2.3 Difference Between Linux and Android

This section discusses the differences between Linux and Android, Linux-based implementations and the differences between x86 and ARM.

Linux vs Android: This study [20] talks about the differences between Linux and Android. The Android OS is structured over Linux and both kernels have striking similarities. However, there are differences in both kernels as many changes have been made to the Linux kernel to build the Android kernel. Several modules have been installed into or removed from the Linux kernel to make modifications. The

authors discussed the changed files, including YAFFS2 (35 Files), a solid-state flash memory chip and Goldfish (44 Files), an Android emulator that executes a virtual CPU. It also includes Bluetooth (10 files), Scheduler (5 files), timed GPIO support, New Android Functionality (28 files) whose subsystems are: IPC Binder that works with higher-level APIs than the standard Linux, Low Memory Killer and Ashmem, which is an Anonymous Shared Memory system. Further changes include RAM Console, Log Device and Android Debug Bridge, all of which help in debugging. Furthermore, other changed files are a better real-time clock (RTC), Miscellaneous Changes (36 files), switch support, Power Management (5 files) which is the most difficult part of getting right on a mobile device and finally, NetFilter. The authors mention that a CPLD controls an Android device's functions as several processes being executed simultaneously inside an integrated circuit can communicate using signals inside the CPLD. This study discusses loadable kernel modules (LKM) and /dev/kmem device access technology. Advanced Configuration and Power Interface (ACPI) and Power Management, are mentioned to manage the Linux system, where the latter is used on the newer systems. However, Android uses Power Manager instead of APM and ACPI. If an application needs a managed device to be powered on, WakeLocks are run through APIs to execute this task. The power management also includes monitoring the battery life of the devices.

The survey [12] also highlights the key distinctions between Linux for desktop, laptop and server systems and Android for mobile devices. To understand Android's architecture thoroughly, the authors looked at it from the bottom-up fashion. Subsequently, broader topics like architecture, core design of the kernel, Java VM, standard C Library, power management and file system were focused on this study for a better understanding of the Specifics of Android's design. Android does not use the standard Linux kernel even though it is based on Linux. Besides, YAFFS, the initial Linux file system that is optimized for NAND flash memory, is used by Android. Meanwhile, flash file systems don't have seeking times like general-purpose disk file systems do. However, they still have lifetime and error correction constraints. Moreover, a faulty file system can quickly cause the system to crash. So, mobile devices that use YAFFS often tolerate significantly fewer file system problems. Also, the authors compared the YAFFS flash file system with Ext3, which has 3 levels of mounting (journal mode, ordered mode and writeback mode). According to the authors, the fundamental differences between YAFFS and Ext3 are file accessibility, block erasing and wear leveling technique.

Linux-based Implementation: To conduct this study [19], the authors developed a measuring framework for Android that consists of general firmware updates and a Linux kernel that has been modified to provide resource utilization data and shows resource utilization data on a modified Linux kernel 2.6.35.7 that are all running on the handheld device.

In this Study [22], the authors implemented Tyche, which is implemented in the Linux kernel 2.6.32. It is operated through the Ethernet and provides operations similar to RDMA without hardware support from the network interface.

x86 vs ARM: The survey [12] works with the target architectures x86 and ARM since both architectures are supported by Linux kernel and Android. The x86 family targets Mobile Internet Devices (MIDs). In contrast, the ARM infrastructure

is more common in handheld devices

3.2.4 Different Implementations of Network Block Device

In this section, we discuss different ways of implementing and building more structures based on NBD, like using GFS, a higher-performance structure built on NBD and reducing protocol overhead by using network storage protocols that utilize NBD.

Usability of NBD Using GFS: This study [3] focuses on GNBD on top of Virtual Interface Architecture (VIA) and later tests the system on Linux-based clusters of PCs. To begin with, GNBD is an enhanced version of the standard Linux NBD that provides simultaneous access by different clients to a single block device at a time. It is different from the standard Linux NBD (which we use in our study) driver as it only works with an isolated client at a time. The intended NBD layer the authors have selected is GNBD. It is because GNBD operating on top of VIA enables the construction of a GFS over system area networks with VIA support. Furthermore, the authors discuss alternative approaches to implementing GNBD, such as simulating the IP layer on top of VIA, employing a Sockets layer from the user side on top of VIA and utilizing a Sockets layer based on the Kernel. From these, they preferred using a Sockets layer on top of VIA on the kernel level, as this approach effectively exploits the VIA's advantages. The KVIPL layer is an important element, allowing kernel codes to utilize the cLAN adapter and the clan1k driver directly. However, an intermediate VCONN (VIA CONNecTion) layer is also introduced as there has to be a notable change in the GNBD core to tweak GNBD directly over the KIVPL. This study measures the bandwidth of file reading and writing using EXT2 and GFS to evaluate GNBD by mounting them on GNBD. Initially, this study expands the KVIPL layer integrated into the VIA driver of Emulex cLAN adapters to provide the identical selection of kernel-level APIs as VIPL. Then, it creates an intermediary layer called VCONN that offers a set of kernel-level interfaces that resemble Sockets over KVIPL. Next, it reduces GNBD's need for code modification while increasing performance using the VCONN layer. The testing was done using TCP/IP over 100Mbps FastEthernet (FE) and LANEVI and VCONN over cLAN and measuring the performance improvement of the file system using Bonnie++. When comparing the performance findings, GNBD/VCONN outperforms GNBD/LANEVI and GNBD/FE regarding read/write bandwidth. Similar to this paper, our study utilizes NBD on the Linux kernel to emulate local storage through a remote server. However, we focus primarily on expanding the storage of Android devices which was not the aim of this study.

Buildability of NBD with Higher Performance: This study [6] focuses on developing and implementing a high-performance networking block device (HPBD) that utilizes InfiniBand fabric. The HPBD is a exchange device that is used by virtual memory (VM) system of the kernel. It makes the page transfers between distant memory servers very efficient. The design of the HPBD is based on the network block device (NBD) framework. The present work presents the concept of remote paging. This technique automatically includes remote memory within the local memory hierarchy, specifically between the main memory and disk. They make a comparison of a design on top of the kernel with a design built on top of the user level, where the kernel-level design is deemed more advantageous than

the other since kernel-level design evades the problems of user-level design, such as pages continuing to be disk pages by the underlying OS, The basic implementation method for the memory protection system having a substantial overhead and user space design only being advantageous to apps using the library and is not entirely application transparent. A kernel module implementation can also lead to portability between several platforms. The kernel-based approach requires the support of asynchronous communication as most OSes do not support preemptive kernel mode. They utilize VAPI's thread safety functionality and the verb interface supplied by their InfiniBand stack and prefer RC service as the network transport.

Network Storage Protocols for Reduced Protocol Overhead:

In this study [22], the authors introduce Tyche, a storage protocol connected on the network that runs on top of an Ethernet connection and offers functionality identical to RDMA, which also does not necessitate hardware assistance from the interface aimed at network connectivity, along with its design, implementation and evaluation. As claimed by the authors, Tyche effectively handles NUMA affinity, pre-allocates memory and can effectively minimize host overheads by mapping Input/Output requests to network messages. To reduce synchronization, it isolates the thread processing context and the primary structures (rings and memory buffers), normally shared over today's protocols. According to the findings of this paper, for sequential reads and writes Tyche can reach up to 6.4 GB/s and 6.8 GB/s, respectively on 6x10 Gbits/s network devices. The findings also demonstrate that NUMA affinity must be considered for maximum throughput and scalability. If not, throughput can decrease by up to 2x. A single packet is utilized by an I/O request/completion message. Tyche uses two distinct queues already assigned, one for data messages (damq) and one for request messages (remq), to lessen the burden of memory management. Tyche picks one connection at the initiating side for every up-to-date Input/Output request. Through this connectivity, it receives a dm-ID in the damq and an rm-ID in the remq. When it comes to the topic of synchronization, multiple threads can send requests at once through the send path and As a result, Tyche synchronizes the access for the NIC as a whole, along with queues and rings. The block layer employs a set of mutexes on top of the initiator for access to remq and damq that is not shared. In the receiving process, a lone thread handles unresolved occurrences and processes the received rings while network threads query NICs simultaneously for events to be received. This reduces synchronization. To minimize overhead regarding the management of memory when transmitting or accepting data files and transmitting Input/Output requests to the receiver device, Tyche employs pre-assigned remq and buffer pools. The pre-defined pages are utilized at the target to send and receive data and issue regular Input/Output requests to the storage device. Network Interface Controller (NIC) positioning on server sockets, kernel Input/Output and Network Interface Controller (NIC) data buffers, protocol data structures, and application buffers are the four components of the I/O path connected to NUMA affinity. The first method used in this experiment, kmem-NIC affinity, assigns every page, data structure of each connection, rings, kernel buffers and NIC pages in a specifically configured node of NUMA. The second of the two is full-mem affinity, which combines affinity at the Input/Output request level with the first method mentioned. The study employs a TCP/IP-based Tyche and the widely used software-only NBD (Network Block Device) for accessing external stor-

age. Two systems, initiators and targets, are connected consecutively using various NICs to form the experimental platform.

3.2.5 Security And Data Privacy

This section talks about the papers that deal with security and data privacy.

This study [17] focuses on data privacy as cloud hosting companies may monitor personal data stored by users in the cloud storages legally or illegally, resulting in privacy concerns. The system RFS allows one user to control access to all the files stored in the cloud and to decide which data they want to encrypt, while the RFS client software gives protection and synchronizes data with the RFS servers across the internet.

The study [24] proposes a Secure Block Device, a solution to ensure data privacy for data at rest. A TA with simple block storage is implemented that uses the SBD method to store blocks. Authenticated Encryption (AE) working with a Merkle-Tree is implemented to this system. SBD supports applications that need quick and reliable random block access to data instead of applications that require a full-scale file system. The SBD method was proposed for Trusted Applications operating with Trusted Execution Environment (TEE) on an ARM TrustZone.

This study [18] suggests significant benefits to breaking up files into smaller segments to transfer files to the server. One of the benefits is allowing users to select multiple servers from various providers. However, they do not have to worry that they will have all the pieces necessary to reassemble the original file. Thus, preventing others from accessing user content, helps safeguard their privacy

3.2.6 Other Topics Regarding NBD, Linux and Android

This section discusses concepts like I/O parallelism, loadable kernel modules and the contrast between user and kernel spaces.

I/O Parallelism: The article [19] proposed a solution to utilize the I/O parallelism already present on the majority of phones. That is a flash drive that is an internal device and an SD card deemed external. The Input/Output operations are stripped to the RAID-0 devices segmented into 4KB blocks; they created a straightforward software RAID driver for Android. the SQLite interface alterations or the programs are the solution source. They propose that a data-centric Input/Output interface may allow the developer to recount the Input/Output requisites involving its coherence, reliability and the characteristics of the data files with no need to be concerned about the storing manner. For instance, a key-value store designed specifically for cache data files does not require super high reliability. It is said that SQLite, as its Web cache, cannot utilize the key-value store cache as effectively as the web browser does.

Loadable Kernel Module : The study [17] talks about the client side of RFS being implemented as an LKM or Linux kernel module and the components that require higher performance (for example, the local cache storage and metadata manager) are implemented into the kernel. The NBD module we implemented in our study is also a Loadable Kernel Module (LKM) similar to RFS.

User Space and Kernel Space: This paper [7] suggests implementing user space applications, which are commonly assigned in the user spaces in the kernel. Applying a network program that works in the Kernel space opens the door to removing the redundant process between the operating system kernel and the user buffer. A total of 3 test scenarios were conducted in this paper, where the first test, a sample code, is altered to achieve the clock ticks needed to execute system calls. For the second and third test scenarios, UDP applications are implemented in kernel and user spaces to evaluate. The results show that implementations in the kernel space decrease CPU load because of the prevention of constantly shifting between the kernel and user spaces. Even after the positive outcomes, there can be risks of security breaches and instability, for which the authors suggest implementing I/O operations within kernel spaces only during critical performance issues.

Distributed Source Sharing: The research work [36] explains distributed resource sharing where the authors have presented an information services architecture that handles the requirements of performance, security, scalability and robustness of the grid technology-based system. Their architecture followed MDS-2, a part of the widely deployed and applied Globus Grid Toolkit. Grid Information service requires few constraints, such as distributing information providers throughout the system, Managing failure and Diversity in information service components. They have implemented the system following MDS-2.1 Protocol Engine, GSI-Single sign-on for security, Information providers GRIS and aggregating the directories with GIIS. This debate made it quite evident that there are many similarities between our GRIS and GIIS systems. An LDAP front end is a protocol processor, authenticating users and filtering results. As part of the Globus 1.1.3 software release [36], their version of this infrastructure, named MDS-2, has been employed in many several ways. They are working hard to make push techniques based on subscriptions and access control strategies that are more complicated.

High-Performance Distributed Systems: According to the research work [38], working with High-Performance Distributed Systems can be challenging for many programmers, as they observe various types of performance problems during the implementation of HPDS. Figuring out what's going on in this complicated system would help understand the performance problems along with finding out ways to characterize the problems. In this paper [38], a method called NetLogger was employed to detect complications in networks and distributed systems. Instead of changing the applications from top to bottom, this performance characterization effort tries to make very speedy components that can be employed as support materials for high-performance programs. This paper also discussed other research projects dealing with network performance analysis. These projects include packages like Pablo, Paradyn and Upshot. In this paper, it is said that the NetLogger approach is unique because it combines network-level monitoring with application-level monitoring. Using NetLogger to find and fix performance problems using real-time data analysis automatically will be studied more in the future.

Hardware as a Service (HaaS): This paper [21] deals with the conceptual basis of using hardware as a service where they have implemented the idea as a middle-ware named Cloud-Disco (Cloud for Distributed Collaboration of Autonomous Organizations). The core idea was to create a distributed cloud for sharing hardware components virtually connected to the machine or embedded systems.

The Cloud-Disco software has three layers: The service layer, which has a Cloud Manager module. Its purpose is to control the overall operation of the local system. The next layer is the middleware layer, divided into a few sub-modules. The Cluster Controller (CM) handles the process of selecting computers or servers that are eager to store their data files in the cloud. The Hardware Provider Host (HPH) can access the locally connected hardware. With the help of Hardware Consumer Host (HCH), the end users can use the resources provided by the upper layers. In HCH, the Host Controller (HC) creates the virtualized hardware for end users. Then, in the last layer, which is the communication layer, the Bus Communication Controller (BCC) or the Direct Communication Controller (DCC) [21] starts according to the needs of the end user, which the HPH or HCH determines. The application of Cloud-Disco was applied in integration testing and system testing. In integration testing, they have emulated specific hardware for a device. It would give more time to the developers to develop the driver for that specific hardware. Furthermore, it also helped to test systems for automobiles' ECU (Electronic Control Unit) section for brakes and speed adaptability with the help of sensors.

The related papers provide valuable insights into NBD's design, performance and applications in various contexts, including virtualization, cloud computing and disk-less systems. They shed light on the benefits and considerations when utilizing NBD for remote storage access, distributed computing and optimizing utilization of resources.

Chapter 4

Methodology & Security Architecture

Linux’s Network Block Device (NBD) protocol is a kernel module that allows access to block devices over a network. The “modprobe” command can load this module, which is present in most Linux distributions [1]. The “nbd-server” command, which enables the user to specify the block device to be exported along with the IP address and port number to listen on, can export a block device over the network after loading the NBD kernel module [3]. The “nbd-client” command lets the user specify the IP address and port number of the NBD server and the local mount point for the block device can be used to access the NBD block device on the client side. Remote machines can access block devices as if physically connected to the local machine once the NBD block device has been mounted and is usable in Linux like any other block device. NBD performs better than iSCSI and can maintain a stable throughput in the case of read and write operations even with the change of numbers of NIC ports [22]. This offers a versatile and effective method of managing storage resources.

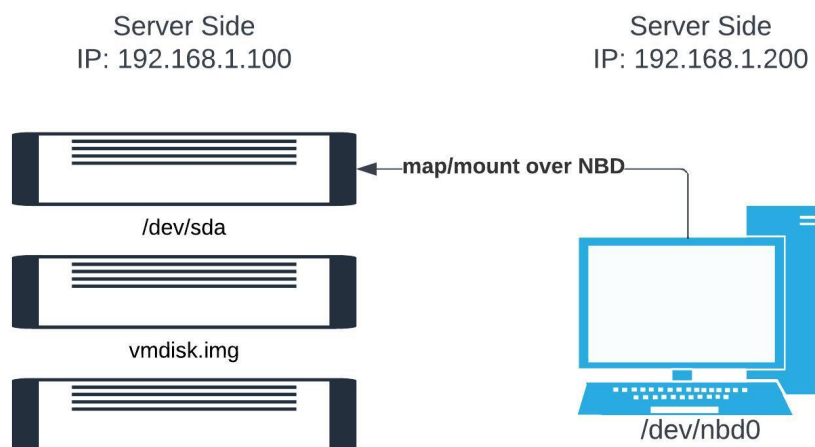


Figure 4.1: NBD Use Case Scenario

Users can resize or create partitions, create a file system (like a local file system) and also create btrfs/zfs/glusterfs storage pools while using a storage device that has been mounted via NBD. Operations from the client on `/dev/nbd0` will have the same effect on the server as they would if you were running them locally with `/dev/sda` as the target as long as the export that the client is using at `/dev/nbd0` is mapped to a device like `/dev/sda` on the server.

Android is built on the Linux kernel [12]. That said, the Android kernel is very similar to the Linux kernel in architecture and functionality. The Unix operating system is the foundation for both kernels, sharing similarities in many ways. The Android kernel is a customized Linux kernel tailored for portable electronics. Process management, memory management and device drivers are just a few similarities between it and the Linux kernel's features and functionalities. Mobile-specific features like power management and support for touchscreens and other input devices are also included.

One of the key similarities between the Android kernel and the Linux kernel is their open-source nature. Both kernels are released under open-source licenses. So, it means their source code is freely available for anyone to view, modify and distribute. This has led to a large community of developers and contributors who work to improve and enhance the functionality of both kernels. Overall, the Android kernel is very similar to the Linux kernel in architecture and functionality. However, it includes additional features and optimizations specific to mobile devices.

The Android platform has been further developed with several libraries and the Linux kernel to enable complex functionality. These libraries were sourced from numerous open-source projects. The Android team created its own C library to deal with licensing issues. The "Dalvik Virtual Machine" is a Java runtime engine they developed that has been optimized for mobile platforms with limited resources. Finally, the application framework was created to provide system libraries to end-user applications concisely.

This methodology provided a comprehensive approach for connecting to an NBD server and accessing the exported block device from a client machine. The installation of necessary packages, connection establishment, block device operations and subsequent disconnection were carried out systematically. Adhering to this methodology enabled successful interaction with the NBD server and utilization of the exported block device on the client side.

4.1 Client-Server Architecture

The NBD (Network Block Device) protocol highlights the client-server architecture within the computer networking domain. A remote server communicating with the client (typically an Android device) running Linux provides the storage resources under the following framework. NBD allows access to data over a network by treating remote block devices as locally attached, providing a highly efficient data access method.

After successfully connecting to the NBD server, the client will begin communicating with it by actively listening for requests. After successfully establishing a

connection between the client and the remote storage, the client can request specific data blocks from the remote storage. After that, the server will obtain the requested data blocks from the client end and send those blocks throughout the network. The benefits are the ability to boot from a diskless image, virtualization support and distributed storage system support. Nevertheless, the NBD protocol uses a client-server architecture, which encourages modularity, scalability and centralized management. Consequently, it is an adaptable option that may be used for a wide variety of storage and data retrieval cases in settings connected to the internet.

4.1.1 Client

The process of integrating the NBD (Network Block Device) device with the Android OS needs a set of procedures aimed at achieving smooth integration and implementing access control measures. An NBD (Network Block Device) device will be eventually connected to the Android kernel space using the NBD client module. This module provides a seamless communication interface between the Android device and remote storage. It enables developers to retrieve and easily manipulate data over a network connection by recompiling the NBD kernel module as LKM and loading it using an on-demand request created by Android Daemon.

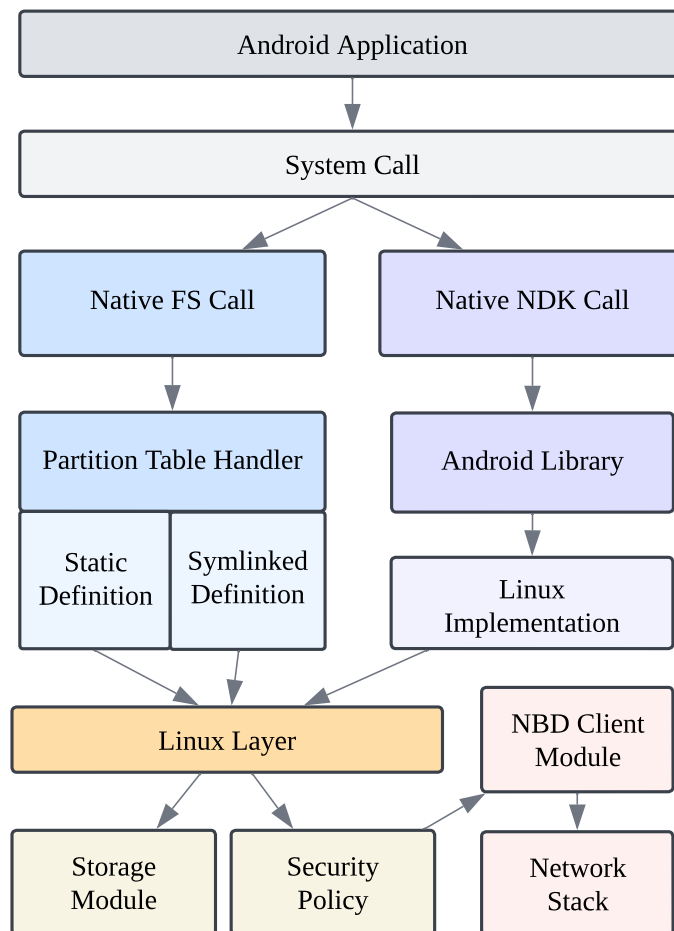


Figure 4.2: Client Architecture

Following the 4.2 figure, after successfully integrating the NBD device into the kernel space, the subsequent task involves mounting it as a directory on the system partition. The system partition is under the “/system/mnt” directory in the Linux layer, where the Android operating system usually houses critical system files and configurations. By mounting the NBD device in this location, it can be integrated into the Android system, enabling it to acquire the essential privileges and access rights required for smooth operation within the OS environment.

A symbolic link (symlink) is created from the system partition to the user data partition in “/etc/fstab” entry to achieve regulated user access to the mounted NBD device. The symlink bridges the NBD device’s contents and the user data area, following the DAC (Discretionary Access Control) definitions. In the Linux and Android development environments, adhering to the principles of DAC (Discretionary Access Control) is essential. The DAC principle ensures that users have the necessary control over their objects. By establishing a connection between the NBD (Network Block Device) and the user data partition, Android users can effectively manage access control for the data stored within the NBD device.

Nevertheless, it is crucial to acknowledge that if the NBD device fulfills a function beyond storage, it can still be accessible to Android users via native system calls using Android NDK. It enables seamless interaction between the Android system and the NBD device’s functionality, circumventing conventional file-based access approaches. The ability of Android to effectively handle NBD devices allows for seamless integration and utilization of various remote resources, such as storage devices or specialized functionalities, in alignment with MAC principles. It greatly enhances the flexibility and utility of the Android operating system.

4.1.2 Server

When exposing a hardware device via a network stack using the Network Block Device (NBD) module, we must design an architecture properly to ensure the data’s integrity, the system’s security and the user’s ability to restrict their access. The steps involved in this process are essential in their own right.

The NBD module establishes the connection between the hardware device and the network. It creates a link between the actual hardware and remote users. We set up the server with multiple layers to protect technology and data. In this setup, we set up an interface with the “/dev” (device) directory to make a virtual version of the actual hardware device. The virtual hardware layer acts as a safety measure by protecting the actual hardware from direct network interactions. It makes it less likely that physical damage will happen.

We set up a security abstraction in the virtual hardware layer. This layer has strict access controls and strong authentication and authorization methods built into it. This method 4.3 improves security by making it harder for unwanted people to get in and making the system more secure.

The integration of a load balancer enables efficient resource optimization and traffic distribution by selecting the most appropriate virtual hardware instance from the pool of newly created virtual hardware devices. Dynamic allocation in Linux and

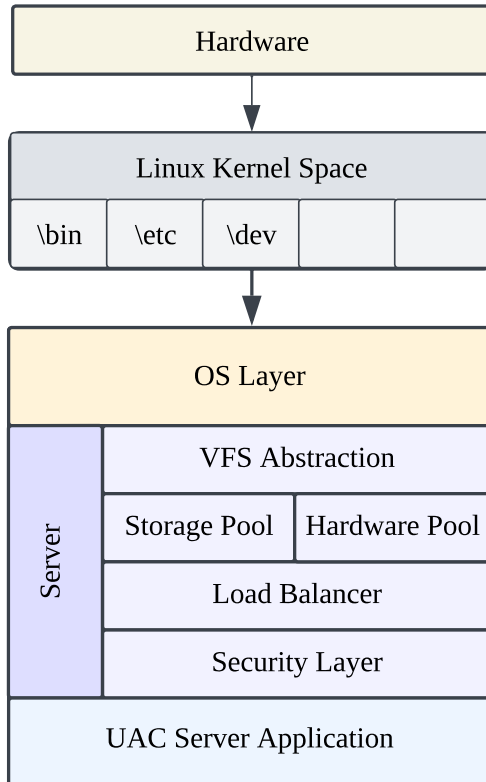


Figure 4.3: Server Architecture

Android development is a crucial aspect that guarantees optimal resource utilization and prevents any individual instance from being overloaded.

We implement a User Access Control (UAC) mechanism to bolster security further. This mechanism enables the controlled exposure of the socket address, granting authorized users or clients access to connect and interact with the hardware device through the NBD protocol while curtailing access for unauthorized users.

Our approach revolutionizes the hardware device into a securely accessible network asset. The layered architecture, security abstractions, load balancing and UAC mechanisms in Linux and Android development collectively guarantee the preservation of data integrity, strict access control and efficient resource management. Remote users can securely establish a connection and engage in interactions with the hardware device, ensuring the preservation of the integrity and security of the underlying physical hardware infrastructure.

4.2 Security Architecture

In designing a security architecture that incorporates both the VFS (Virtual File System) layer for physical device damage protection and TLS (Transport Layer Security) with NBD (Network Block Device) for data security, we can create a robust and layered approach to safeguarding sensitive information. Let's explore this architecture 4.4 in detail.

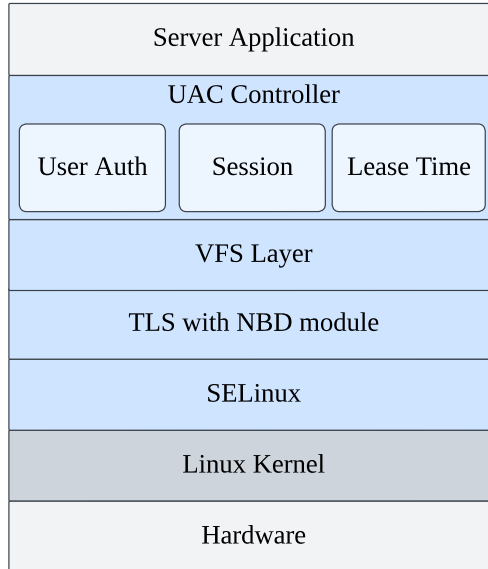


Figure 4.4: Security Architecture

- Physical Device Damage Protection with VFS Layer:** The VFS layer acts as an abstraction layer between the operating system and various file systems, providing a unified interface for file operations. In physical device damage protection, a virtual device node must be created and provide clear instructions on sanitization to implement additional security measures. For example, abstraction tools like “dd” can create an image of the original block-level storage device to represent a virtual storage device. On the other hand, tools like “mknod” can be used to create a virtual device file representation of the actual one to implement the Security Policy Language (SPL) in a logical approach by using DAC or MAC. Here are some key components of this architecture:
 - Filesystem Encryption:** The VFS layer can support encryption mechanisms to ensure that data stored on the physical device remains protected even if the device is compromised. This can involve encrypting file data at rest using strong encryption algorithms, making it unreadable without proper decryption keys.
 - Access Control:** The VFS layer can enforce access control policies to limit unauthorized access to sensitive files. This includes implementing file permissions, user/group-based access restrictions and auditing mechanisms to track file access and modifications.
 - Filesystem Integrity:** By implementing checksums or digital signatures, the VFS layer can verify the integrity of files on the physical device. This helps detect tampering or unauthorized modifications to the files, providing additional protection against physical attacks.
- Data Security with TLS and NBD:** To ensure secure communication and data protection over the network, TLS can be used with NBD. This combination establishes a secure channel for transmitting data between the NBD

client and server. TLS is natively supported in NBD [3]. Here's how this architecture can be implemented:

- **TLS Encryption:** TLS encrypts data in transit, preventing eavesdropping and unauthorized access to sensitive information. By configuring the NBD server and client to utilize TLS, all data transmitted over the network is encrypted, ensuring its confidentiality.
 - **Certificate-based Authentication:** TLS utilizes digital certificates for authentication. The NBD server can be configured with a trusted certificate and the client can verify the server's identity using the certificate's public key. This mutual authentication ensures the integrity of the connection and protects against man-in-the-middle attacks.
 - **Data Integrity:** TLS also ensures data integrity by using Cryptographic algorithms, such as message authentication codes (MACs), to detect any tampering or modification of the transmitted data. This guarantees that the data received by the NBD client is the same as what was sent by the server, assuring data integrity.
3. **UAC and Session Management:** A User Access Control (UAC)-enabled server architecture is a vital part of handling NBD (Network Block Device)-exposed devices well, giving users precise control over their access and making session management easier. This robust architecture ensures that only authorized people can get customized access to NBD devices, making the best use of resources and improving security. With UAC in place, you can set up fine-grained access rules that say which users can connect to which NBD devices, what they can do and for how long. This level of control is very important for businesses because it lets them build flexible service models around NBD that can be changed to fit their needs. Companies can offer different access plans and charge users based on how much they use, how much storage space they need or how long their sessions last. Thus, a thriving business ecosystem can emerge, providing diverse services and monetization opportunities in NBD while ensuring that data remains secure and accessible only to those with the appropriate permissions.
- **Granular User Access Control:** UAC allows for fine-grained control over user access to NBD devices. Server administrators can define specific permissions and restrictions for each user or group, ensuring only authorized individuals can connect to and interact with particular NBD resources. It enhances the security and data protection of access control.
 - **Session Management:** UAC also facilitates efficient session management for NBD connections. Administrators can set session timeouts, ensuring that users are automatically disconnected after a defined period of inactivity. It helps to make the most of the system's resources and prevents all unauthorized access due to forgotten or unattended sessions, making the system more efficient.
 - **Customized Service Plans:** Businesses can make service plans that fit the needs of each customer with UAC. They can give different levels of access, storage space, or data transfer rates, which makes it possible

to use different ways to make money. This flexibility allows the creation of services based on subscriptions, pay-per-use models, or tiered pricing systems, which opens up more ways to make money.

- **Auditing and Logging:** UAC architectures have tracking and logging features. So, every time a user interacts with an NBD device, it records it. This makes it possible to see an entire history of access and operations. These logs are helpful for compliance, fixing and security analysis, ensuring everything is precise and everyone is responsible.
- **Scalability and Resource Optimization:** As users grow, computers with UAC can quickly scale up to meet the growing demand. Administrators can give resources to users based on their entry levels and needs. This makes sure that the resources are used in the best way possible. This improves the user experience and allows companies to change and grow their NBD offerings as their customer base changes.

By implementing this comprehensive security architecture, Android devices can benefit from enhanced data security and privacy. Combining the VFS layer's physical device damage protection mechanisms and TLS with NBD's secure communication ensures that sensitive information is protected against unauthorized access, tampering and interception. This robust security framework establishes a strong foundation for secure data sharing and storage within Android devices, fostering trust and confidence in their ability to handle sensitive data securely.

Chapter 5

Implementation

5.1 System Specification

The experimental setup consists of a server, a network device and a client. Here are the hardware components used in each part:

5.1.1 Server

- **Processor:** Intel Core i3-8130U. This is a dual-core, four-thread processor commonly found in laptops. It has a base clock speed of 2.2 GHz, up to 3.4GHz in turbo and supports hyper-threading.
- **RAM:** 12GB. The server is equipped with 12 gigabytes of random-access memory (RAM), which provides temporary storage for data and program instructions.
- **Storage:** 128GB M.2 SATA III SSD. The server utilizes a solid-state drive (SSD) with a capacity of 128 gigabytes. The M.2 form factor and SATA III interface ensure fast data transfer rates and efficient storage performance.
- **Network Connectivity:** The server is connected to the network using a local area network (LAN) connection. We have used an IEEE 802.11 b/g/n wireless PCIe NIC card as the network interface.

5.1.2 Network Device

- **Archer C20 Router:** We have used the Archer C20 Router as the network device in the setup. It has an IEEE 802.11 ac-2013 with a 2.4 GHz and 5GHz bandwidth network interface. It is a consumer-grade wireless router commonly used in small environments.

5.1.3 Client

- **Processor:** We used an Intel Core i7-8750H which has six-core, twelve-threads. It has a base clock speed of 2.2 GHz, can go up to 4.1GHz in turbo, and supports hyper-threading.

- **RAM:** 16 GB. The configured server is equipped with 16 gigabytes of RAM.
- **Storage:** 128GB M.2 SATA III SSD. The server utilizes a solid-state drive (SSD) with a capacity of 128 gigabytes.
- **Network Connectivity:** The server is connected to a local area network (LAN) connection. It is an IEEE 802.11 b/g/n wireless PCIe NIC card.

Overall, this experimental setup portrays a server with an Intel Core i3 processor, 12GB RAM and a 128GB SSD connected to the network via LAN. The client has an Intel Core i7 processor, 16GB RAM and 128GB SSD via Wi-Fi. An Archer C20 Router is used as the network device.

5.2 Configuring the Server

The methodology section outlines the detailed steps to set up an NBD server on Ubuntu 23.04. The objective was establishing a server configuration capable of exporting block devices over the network. The following procedure 5.1 explains the process of installing and configuring the NBD server, ensuring its automatic startup and verifying its status:

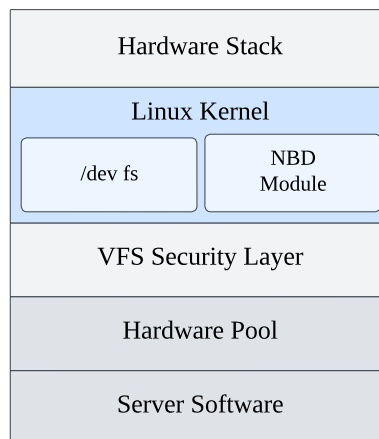


Figure 5.1: Implemented Server Architecture

5.2.1 Package Installation

The initial step involved updating the package repositories to ensure the latest software versions were available. The command is executed to perform the update.

```
sudo apt update
```

Following that, the “nbd-server” package is installed from the official Ubuntu 23.04 repository using the command

```
sudo apt install nbd-server
```

5.2.2 NBD Server Configuration

The configuration file for the NBD server is accessed and modified to define the exported block devices and specify access permissions. The file is located at

```
/etc/nbd-server/config
```

The command was used to open the configuration file in the nano text editor.

```
sudo nano /etc/nbd-server/config
```

This section was identified within the configuration file. This section provides the opportunity to define the block devices that are exported. The line “export-name=exportdevice” was uncommented and updated to define an export. For instance, the line “/dev/sdb” referred to the block device that would be exported. Using the exportname argument, the desired export name, such as “nbd1” was also assigned. It was ensured that the block device’s right path on the server was provided; therefore, care was taken with it. After the changes were made, the configuration file was saved by hitting “Ctrl + X,” then “Y,” and finally “Enter”.

5.2.3 Automatic Startup

To ensure the NBD server starts automatically during system boot, it can be converted into a systemd unit using the command

```
sudo systemctl enable nbd-server
```

was executed. This command added the necessary system startup links for the NBD server service.

5.2.4 Server Start

The NBD server was initiated using the command

```
sudo systemctl start nbd-server
```

This command started the NBD server service, allowing it to listen for incoming connections.

5.2.5 Server Status Verification

The status of the NBD server and the exported block devices were verified using the command

```
sudo nbd-server -s
```

This command summarized how the server was running, including information about which block devices were exported and what their export names were. With this command, you could ensure the NBD server had started up properly and that the exported block devices were set up correctly.

With this all-in-one method, setting up an NBD server on Ubuntu 23.04 was easy. The right packages had to be installed, the NBD server had to be set up, automatic startup had to be turned on, and the server’s state had to be checked to ensure it worked. This method ensured block devices could be sent over the network without damage.

5.3 Configuring Client

The methodology part explains how to use a client machine to connect to an exported block device. This is done using the architecture shown in the picture 5.2. The steps explain how to install the necessary client packages, how to connect to the NBD server, and how to work with an exported block device:

5.3.1 Package Installation

The next step was to install the appropriate packages on the client system. The command installs the “nbd-client” package, which has the necessary tools to connect to the NBD service.

```
sudo apt install nbd-client
```

5.3.2 Client Connection

To get access to the exported block device and successfully connect to the NBD server, you need to run the “nbd-client” tool. The format of the order was as follows:

```
sudo nbd-client <server_ip> <export_name> /dev/nbd0
```

The IP address or username of the NBD server was put in place of the <server_ip>. The placeholder <export_name> was replaced with the exact export name set up on the server. This made it possible for the client to find the desired block device. “/dev/nbd0” showed which local device on the client machine would be used to access the exported block device.

5.3.3 Block Device Operations

Various operations could be performed once the block device was successfully mounted on the client. For instance, the “mount” command could mount the block device to a designated directory. The following command demonstrates this operation:

```
sudo mount /dev/nbd0 /mnt
```

In the command above, “/dev/nbd0” represents the exported block device, while “/mnt” is the target directory for mounting. Adjustments to the mount point may be made as per the specific requirements.

5.3.4 Unmount and Disconnect

When the usage of the block device is complete, it needs to be unmounted before disconnecting from the NBD server. The ‘umount’ command facilitates the unmounting process:

```
sudo umount /mnt
```

In the command above, “/mnt” refers to the mount point where the block device was previously mounted. After unmounting, it is crucial to disconnect from the NBD server using the “nbd-client” command:

```
sudo nbd-client -d /dev/nbd0
```

The `-d` option specifies the termination of the NBD client connection, ensuring a clean disconnection from the NBD server.

5.3.5 Waydroid

To harness the potential of NBD from the Linux Layer to the Android Layer, we need to install Waydroid on our Ubuntu machine. First, we must install “curl” & “ca-certificates” if not present.

```
sudo apt install curl ca-certificates -y
```

After that, we have to add the repository.

```
curl https://repo.waydro.id | sudo bash
```

Finally, we have to install Waydroid using the following command.

```
sudo apt install waydroid -y
```

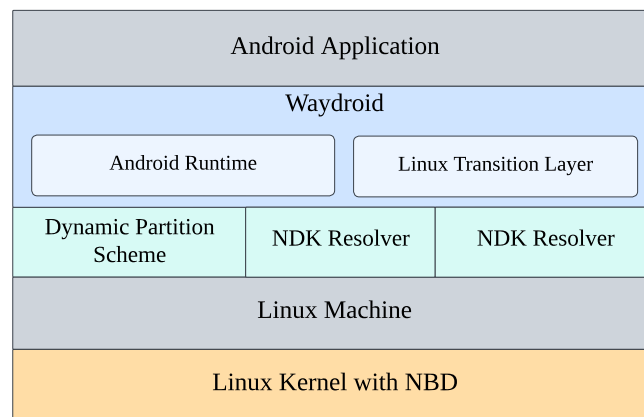


Figure 5.2: Implemented Client Architecture

5.3.6 Symbolic Linking for Dynamic Partition

To use the NBD device inside Waydroid’s Android Runtime environment, a Symlink must be created between the NBD storage mount point and Waydroid’s system partition. The rest of the steps will be aligned according to the methodology.

To create the Symlink, the command is,

```
ln -s /mnt/nbd-mount-point /waydroid-rootfs/mnt/sdcard/mydata
```

To verify the Symlink, we need to use the following command,

```
ln -l /waydroid-rootfs/mnt/sdcard/mydata
```

Chapter 6

Experimental Evaluation

The primary objective of this paper is to comprehensively evaluate the performance of Network Block Devices (NBD) in Android devices when connected to a server. We specifically focus on utilizing Android devices as the client device.

To address the persistent issue of limited storage capacity in Android devices, we propose the inclusion of NBD as a default feature, supplemented by a translation layer that facilitates communication between NBD and various storage protocols. We compare NBD with several prominent storage solutions for our evaluation, including the HTTP translation layer, SFTP, NFS, SMB, Google Drive, Dropbox, OneDrive, GCP and local storage.

We look at a wide range of performance metrics to determine how well these storage options work. These metrics include the read sector, write sector, write file system, build file system, change position table, mount ability, and random access level. By looking at these metrics, we hope to learn more about how each storage option works, how reliable it is, and how fast it is in different situations.

Through our study, we hope to show that adding NBD to Android devices is possible and could have benefits. We compare NBD to the other storage solutions listed above and show how it is better regarding read and write speeds, file system operations, data management, and general performance. This study will help us make a strong case for making NBD a default feature in the Android kernel. This would solve the low storage problem and make it easier for Android devices to connect to network-attached storage (NAS).

Using the Android device as a client, we make a useful and real-world evaluation setting. The results of this study will help Android become a more powerful NAS client that can work well with a wide range of storage solutions. Ultimately, we want to discover things that will help Android devices get better at storage, user experience, and general performance.

6.1 Configuration for Performance Test

A thorough testing method was used to determine how well the NBD communication route worked and how well it worked. The tests included measuring the speed of

read sectors, write sectors, read file system (FS), and write file system (FS) and testing the ability to mount and do random access on the NBD communication channel. The following actions were taken:

6.1.1 Preparation

1. NBD server was set up in the methodology.
2. NBD client was set up in the implementation.
3. NBD server and client were connected over the network to establish communication.

6.1.2 Bandwidth Test for Read Sectors

The bandwidth of reading sectors from the NBD communication channel was measured by executing the command

```
sudo dd if=/dev/nbd0 of=/dev/null bs=1M count=1000
```

6.1.3 Bandwidth Test for Write Sectors

The bandwidth of writing sectors to the NBD communication channel was measured using the command:

```
sudo dd if=/dev/zero of=/dev/nbd0 bs=1M count=1000
```

6.1.4 File System Test for Read FS

1. A mount point directory was created on the client machine

```
sudo mkdir /mnt/nbd
```

2. The NBD communication channel was mounted to the created mount point

```
sudo mount /dev/nbd0 /mnt/nbd
```

3. Data was read from the mounted NBD file system using various file manipulation commands, such as **ls**, **cat**, or **cp**. For example

```
ls /mnt/nbd
cat /mnt/nbd/file.txt
cp /mnt/nbd/file.txt /tmp/
```

6.1.5 File System Test for Write FS

1. A test file was created on the client machine for writing to the NBD file system. For instance:

```
echo "Test data" > /tmp/test.txt
```

2. The test file was copied to the mounted NBD file system:

```
cp /tmp/test.txt /mnt/nbd/
```

6.1.6 Mount and Random Access Test

Random access operations, such as reading and writing specific sectors or files, were performed on the NBD communication channel. The response time for each operation was measured. For example, the following commands were executed to read and write specific sectors:

```
sudo dd if=/dev/nbd0 of=/dev/null bs=1M skip=1000 count=10
sudo dd if=/dev/zero of=/dev/nbd0 bs=1M seek=2000 count=10
```

6.1.7 Cleanup

The NBD file system was unmounted with the command:

```
sudo umount /mnt/nbd
```

The client disconnected from the NBD communication channel, concluding the test procedure.

By following this comprehensive methodology, the performance and functionality of the NBD communication channel were thoroughly examined.

6.2 Comparative Analysis

In the part of the study that deals with contrasts and evaluations, we have divided it into three subsections for clarity. Each of these subheadings aims to shed light on a particular facet of the overall functionality and performance of the Network Block Device (NBD) solution that is of particular importance. These three subsections comprehensively compare how the NBD compares against various storage solutions such as Google Drive, Dropbox, OneDrive and GCP.

Table 6.1 describes the storage ability of various storage solutions in terms of low-level access. In this comparison of data storing and cloud services, we looked at the features of Google Drive, Dropbox, OneDrive and Google Cloud Platform (GCP). Google Drive and GCP showed how powerful they are by letting users read and write sectors, create file systems, change partition tables and use certified services, among other things. They were also great at mounting and random access. On the other hand, Dropbox and OneDrive showed off their strengths by confirming that these tools are there. This comparison shows how different the features of these cloud services are. It also shows the importance of choosing a provider that fits your storage and control needs.

Table 6.2 houses bandwidth comparison test reports among NBD and other storage solutions.

Feature	NBD	Google Drive	Dropbox	OneDrive	GCP
Read Sectors	✓	x	x	x	✓
Write to Sectors	✓	x	x	x	✓
Create File System	✓	x	x	x	✓
Alter Partition Table	✓	x	x	x	✓
Mount Operations	mount	ocamlfuse	rclone	rclone	google-cloud-cli
Random Access	mount	partial	partial	partial	partial
Verified	✓	✓	✓	✓	✓

Table 6.1: Storage Capability Comparison

Cloud Storage	Data Read		Data Write	
	Bandwidth (MB/s)	Time (second)	Bandwidth (MB/s)	Time (second)
NBD	26.9	30.29	26.9	30.29
Google Drive	10.2	100.39	8.7	119.08
Dropbox	7.8	131.28	7.1	143.66
OneDrive	10.9	93.81	8.5	120.24
GCP	10.4	97.73	9.3	109.94

Table 6.2: Bandwidth Comparison

6.2.1 Storage Capabilities

We examine how much storage NBD has compared to what other options can do. In this part of the piece, we will look at several topics, such as the speed at which data can be sent, the reliability of the saved data and the ability to handle vast amounts of data. The file-level storage methods that HTTP, SMB and SFTP use are evaluated based on how they affect the system’s speed. The block-level storage method NBD uses is evaluated based on how it affects the system’s efficiency.

Read Test: In the figure 6.1, we notice a huge difference between NBD and other cloud services. In our test, NBD can hit 26.9 MB/s, as the average speed of other cloud services was around 10 MB/s. We can see NBD performed twice as well as other cloud Services.

Figure 6.2 refers to the duration it took to write the data for NBD and other cloud services. As the speed of NBD was more than twice that of traditional cloud services, it only took around 1/3rd of the time (30.29s) compared to others.

Write Test: In this comparison graph 6.3, the NBD can be consistent like it is in the case of reading data. Here, it performed at a similar speed (26.9 MB/s) of reading. However, the other cloud services could not maintain the consistency of speed; however, they have an average speed of 8.35 MB/s, which is decent enough.

Regarding write duration, figure 6.4 concisely portrays the difference. It took 30.29 seconds for NBD to write the data into the disk, whereas the other services’ speed varied from 109.94 seconds to 143.66 seconds, which is almost 4 times higher than NBD if we take the mean speed them.

6.2.2 Power Consumption

As part of providing data, we look into how fast NBD uses system resources, especially power. This analysis 6.3 considers the effect on mobile devices’ battery life

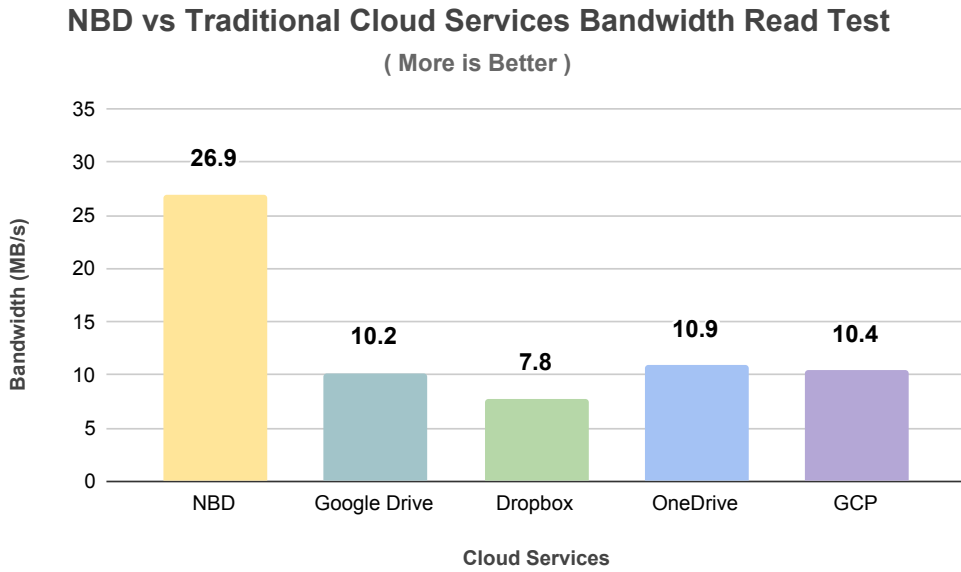


Figure 6.1: NBD vs. Traditional Cloud Services Bandwidth Read Test

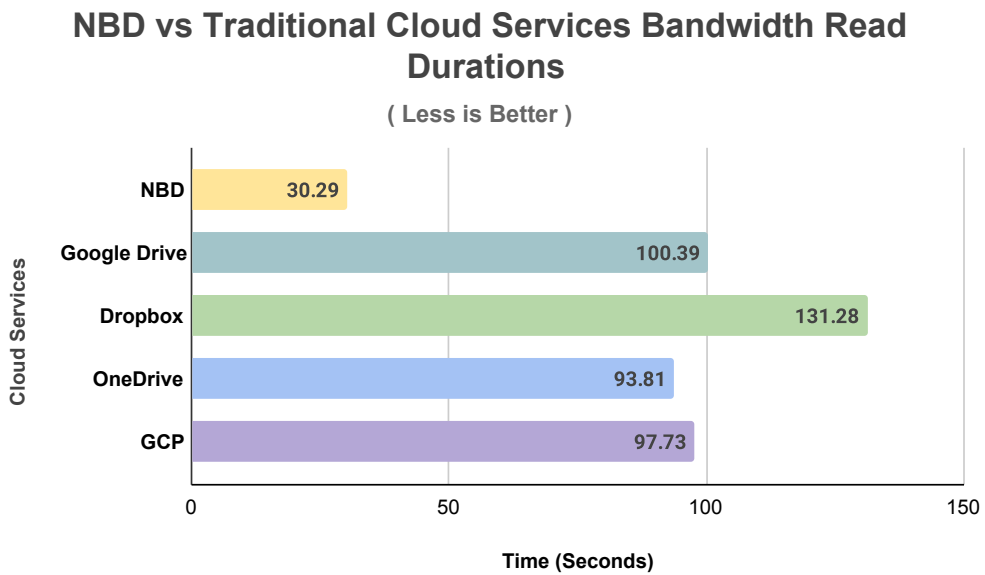


Figure 6.2: NBD vs. Traditional Cloud Services Bandwidth Read Duration

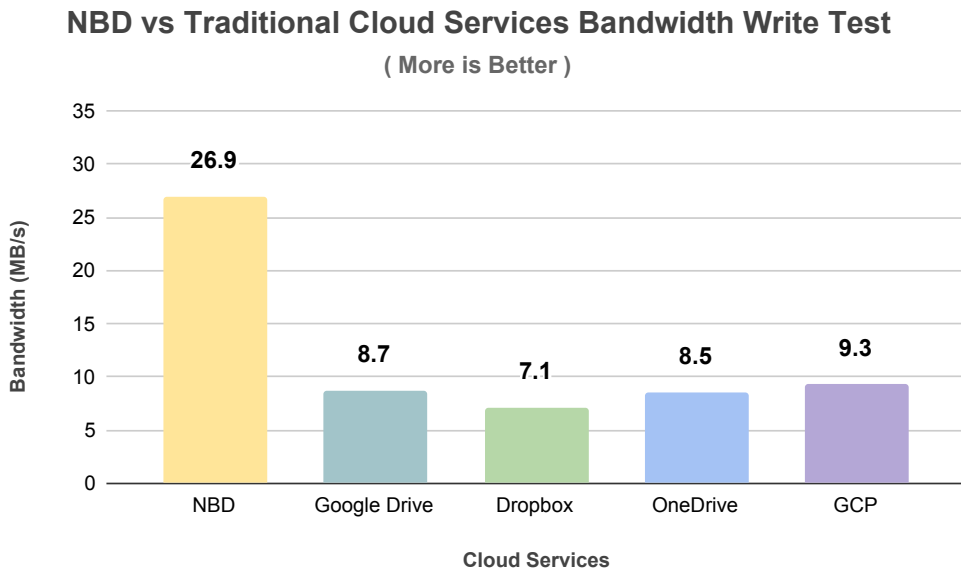


Figure 6.3: NBD vs. Traditional Cloud Services Bandwidth Write Test

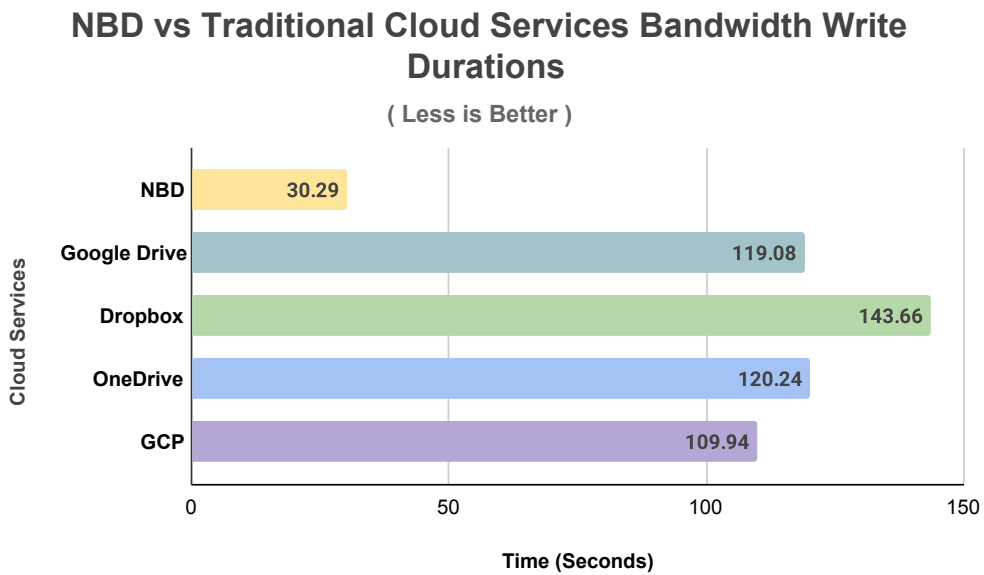


Figure 6.4: NBD vs. Traditional Cloud Services Bandwidth Write Duration

NBD vs Traditional Cloud Service Power Consumption of Read/Write Operation

(Less is Better)

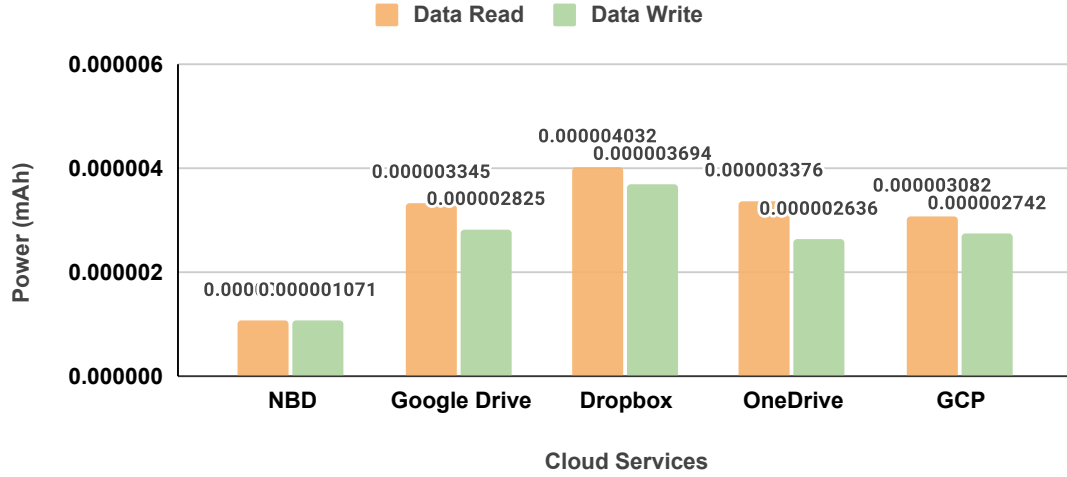


Figure 6.5: NBD vs Traditional Cloud Service Power Consumption of Read Write Operation Test

and NBD's total energy efficiency compared to other protocols by using the formula

$$\text{Power Consumption} = (u_1 - u_0 + \Delta E)$$

Here, u_0 is the power reading before the operation, u_1 is the power reading after the operation and ΔE is time (second * idle power consumption rate within 1 second and t is the time required to perform the measurement operation.

Cloud Storage	Data Read		Data Write	
	Power (mAh)	Time (second)	Power (mAh)	Time (second)
NBD	0.000001071	30.29	0.000001071	30.29
Google Drive	0.000003345	100.39	0.000002825	119.08
Dropbox	0.000004032	131.28	0.000003694	143.66
OneDrive	0.000003376	93.81	0.000002636	120.24
GCP	0.000003082	97.73	0.000002742	109.94

Table 6.3: Power Comparison

To visualize the power consumption better, even though it is a very small amount, we plotted it in figure 6.5. We can see that the NBD consumed much less power than other competitors to perform similar operations in the same environment.

6.2.3 Cost Structure

The cost structure figure6.4 of installing NBD and other storage solutions is evaluated and analyzed. This analysis considers direct and indirect expenses, including requirements for hardware, license fees (if required), maintenance and scalability

Category	Storage (GB)	Price (USD)
iPhone 15	128	799
	256	899
	512	1099
iphone 15 Pro Max	128	1199
	256	1299
	512	1499
	1024	1699
NBD	128	281.71
	256	291.71
	512	302.71
	1024	322.71

Table 6.4: Cost Comparison

considerations. Our goal is to shed light on whether or not the implementation of NBD is financially viable in various contexts.

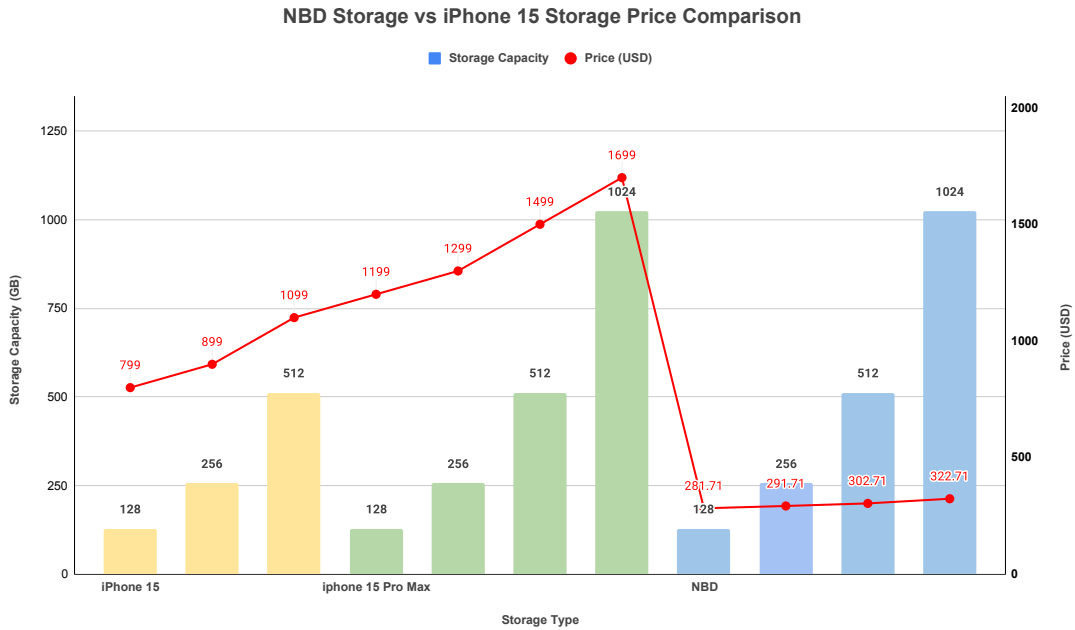


Figure 6.6: NBD Storage vs iPhone 15 Storage Price Comparison

Figure 6.6 suggests the local storage base solution price increases proportionally with storage increment while the NBD solution price has a less tendency to go higher even when the storage capacity exits the equilibrium point.

Our goal is to present a comprehensive and organized analysis of the NBD's performance and suitability in comparison to other storage protocols that are widely used. We have broken down our evaluation into these three subsections to achieve this. This method allows us to comprehend how NBD may cater to particular storage requirements and correspond with goals, regardless of whether they place a higher value on speed, energy efficiency or cost-effectiveness.

Chapter 7

Open Discussion & Challenges

7.1 Theoretical Comparative Study

Android devices can potentially serve as Network-Attached Storage (NAS) clients, offering various use cases and notable benefits. Let's examine these use cases and benefits in-depth, using technical terms to illustrate their significance:

1. **Media Streaming:** Android devices can act as NAS clients by connecting to NAS servers and streaming media content stored on the network. With support for protocols such as DLNA (Digital Living Network Alliance) or UPnP (Universal Plug and Play), Android devices can access media files hosted on NAS servers and play them seamlessly on compatible media players or streaming applications. This lets users enjoy high-quality media content directly from their Android devices, leveraging the NAS server's storage capacity and network capabilities.
2. **File Synchronization and Backup:** Android devices can synchronize files with NAS servers, allowing for seamless data backup and access across multiple devices. Android devices can use protocols like rsync or Syncthing to maintain file consistency between local and NAS storage, ensuring data integrity and availability. This capability is particularly useful for professionals or individuals who require continuous synchronization of files across devices, providing a reliable backup solution and mitigating the risk of data loss.
3. **Remote Access and File Sharing:** Android devices can establish secure connections with NAS servers, enabling remote access and file-sharing functionalities. Android devices can securely connect to NAS servers outside the local network using protocols such as SSH (Secure Shell) or VPN (Virtual Private Network). This allows users to access and manage files stored on the NAS remotely, facilitating collaborative work environments and seamless file sharing among authorized users.
4. **Data Caching and Offline Access:** Android devices can utilize NAS storage as a cache partition, optimizing data access and enabling offline access to frequently used files. By implementing technologies such as NFS (Network File System) or SMB (Server Message Block), Android devices can store frequently accessed files locally in a cache partition, reducing latency and network

overhead. This feature is beneficial for applications that rely on quick access to data, such as media players or productivity apps, providing smooth offline access to essential files.

5. **Distributed Computing and Processing:** Android devices can leverage NAS storage to offload computationally intensive tasks or utilize distributed computing frameworks. By utilizing protocols such as SSHFS (SSH Filesystem) or NFS, Android devices can access remote storage resources and perform complex computations or data processing tasks. This distributed computing capability allows Android devices to tap into the vast storage capacity of NAS servers, enabling resource-intensive applications such as image recognition, machine learning, or distributed data analysis.

Traditionally, the NAS (Network Attached Storage) solutions available in Android often come with client-server apps that aren't open source, which means it is limited to a certain environment. For example, platforms like Plex, Jellyfin, DLNA, and UPnP require users to use special software to view and control their files and media. This closed, vendor-dependent method can make things less open and fluid.

Network Block Device (NBD) storage options, on the other hand, offer a cross-service architecture that lets users use their favorite apps for different services. NBD works at the block level, making it a more neutral and flexible way to store and view data. With NBD, users aren't tied to a specific software environment. Instead, they can choose the apps that work best for them.

However, this flexibility goes beyond streaming video and managing files. NBD can be used for various tasks, such as synchronizing data, backing it up, letting people view it from afar, and doing distributed computing. Users can set up NBD to fit their needs, ensuring their favorite apps work well with their storage options. In effect, NBD frees users from being locked into one vendor, which makes network storage and data management more open and flexible.

7.2 Challenges

While this paper aims to evaluate the performance of Network Block Devices (NBD) in Android devices and provide insights into their potential benefits, it is essential to acknowledge certain limitations that may impact the scope and generalize ability of the findings. These limitations include:

7.2.1 Hardware Dependency

Due to consumer-level Android handheld devices as a non-developer-friendly manufacturing architecture, it is much harder to implement NBD without OEMs' proper licensing agreement. Therefore, to test the functionality and tap the potential of the Android ecosystem, we had to implement the Android operating system layer with Waydroid on top of a generic x86 Linux Kernel to imitate the functionality of an Android device. However, it's important to note that different Android devices may have varying hardware configurations, potentially affecting performance results. The findings may not directly translate to all Android devices, and further testing on a wider range of hardware is warranted.

7.2.2 Software Compatibility

The evaluation focuses on the performance of NBD with an x86 Linux server. However, efforts are made to ensure compatibility; variations in software versions, device drivers, and kernel configurations may influence the results. The outcomes may differ when using different server configurations or alternative operating systems, which should be considered in future studies.

7.2.3 Limited Comparison Candidates

While this paper compares NBD with popular storage solutions such as SFTP, NFS, SMB, Google Drive, Dropbox, OneDrive, GCP, and local storage, numerous other storage protocols and services are available. The selection of comparison candidates may not encompass the entire landscape of NBD's horizon storage options, potentially limiting the breadth of the evaluation.

7.2.4 Performance Metrics Selection

The chosen performance metrics, such as read sector, write a sector, read file system, write file system, create a file system, alter position table, mount ability, and random access level, provide valuable insights into NBD performance. However, other metrics, such as latency, data transfer rate, and concurrency, may also contribute to a comprehensive assessment of NBD's performance. Future studies could consider incorporating additional metrics for a more holistic evaluation.

7.2.5 Network Environment Variability

The network environment, including network congestion, latency, and bandwidth limitations, can influence the performance of NBD and other storage solutions. The evaluation in this paper assumes a relatively stable and optimal network environment. However, real-world network conditions can vary, and the performance of NBD may be affected accordingly.

7.2.6 Limited Generalizability

The evaluation in this paper focuses on a specific scenario involving NBD, a Waydroid-based Android client, and an x86 Linux server. The findings and conclusions may not directly apply to other configurations, devices, or use cases. The generalized ability of the results should be considered within the specific context of the evaluation.

By acknowledging these limitations, future research can aim to address these concerns and expand upon the findings presented in this paper. A more comprehensive understanding of NBD's performance in various scenarios can be achieved through further investigations and wider testing.

Chapter 8

Conclusion & Future Prospects

8.1 Future Prospects

Despite the limitations mentioned earlier, utilizing Network Block Device (NBD) in Android devices offers the potential to share storage mediums and other block devices. This can enable a wide range of applications and expand the capabilities of Android devices. Here are some examples of block devices and potential future research directions to make them compatible with Android:

1. **Network-Attached Storage (NAS):** In addition to sharing storage resources, NBD can be extended to share NAS devices, allowing Android devices to access and utilize remote file systems and network shares. Future research can focus on developing NBD-compatible drivers and protocols for popular NAS devices, ensuring seamless integration and maximizing the benefits of NAS in Android.
2. **Printers and Scanners:** Block devices such as printers and scanners can be shared with Android devices through NBD. This would enable Android users to directly connect to and utilize these devices, expanding printing and scanning capabilities. Future research can explore the development of NBD drivers and protocols specific to printer and scanner devices, facilitating seamless connectivity and efficient usage.
3. **Input/Output Devices:** NBD can potentially be extended to share input/output block devices, including keyboards, mice, game controllers, and more. This would allow Android devices to utilize a wider range of peripheral devices, enhancing user interaction and enabling diverse applications. Future research can investigate the development of NBD-compatible protocols and drivers for different input/output devices, considering device-specific requirements and compatibility.
4. **Block-level Virtualization:** NBD can serve as a foundation for block-level virtualization in Android devices. This would enable the creation and management of virtual block devices representing physical devices or partitions. Future research can focus on developing NBD-based virtualization frameworks and tools, allowing Android devices to utilize virtual block devices with enhanced control and flexibility efficiently.

5. **IoT Devices:** With the proliferation of the Internet of Things (IoT) devices, integrating NBD support for various IoT block devices can enable Android devices to communicate and interact with a wide range of IoT devices seamlessly. The development of NBD drivers is crucial to ensure compatibility and interoperability with Android Devices. And for popular IoT platforms, future research can bring improvements to them.

Integrating the Android through NBD can bring specific benefits. Such as printers and scanners with Android devices would enhance productivity and convenience. It will also enable seamless control and monitoring of connected devices. But it is only possible if we can research more on this technology. A few fields namely NBD-compatible drivers, protocols, and frameworks for block devices are needed to focus on for future research aspects. It will also need to consider device-specific requirements, compatibility and standards. This would bring the whole community of Android developers, device manufacturers and open source community to improve and improvise new possibilities for Android devices.

8.2 Conclusion

In conclusion, this paper has assessed and thoroughly evaluated the performance of Network Block Devices (NBD) in the Android ecosystem. We are primarily focusing on making use of Waydroid to connect client devices to an x86 Linux server. Even though there are some limitations, we have found more potential and benefits of incorporating NBD as a default feature in Android devices.

This paper further acknowledges the extensive potential of network block devices (NBD) to not only open the door to storage sharing but also empower Android devices to act as middleware between remote block devices and the users themselves. By utilizing NBD to its full potential in the Android ecosystem, a connection for smooth communication and control can be created between the users and various devices, such as network-attached storage (NAS), IoT devices, input/output devices, scanners, printers and more.

With this prospect, researchers in the future will be able to explore the development of frameworks, drivers, and other protocols that are compatible with NBD and further make a connection with Android devices to NBD and effectively communicate and interconnect with an extensive range of block devices. This would allow Android devices to serve as an ingenious mediator, resulting in users having more authority over block devices that require remote management and access.

A lot of possibilities for data management, and device integration can be enhanced significantly by enabling NBD in Android. It is also necessary to work on the compatibility of diverse block devices. This not only enhances the capabilities of Android devices but also opens up new avenues for innovation in various domains. Overall, this research paves the way for future advancements in Android with its storage upgrade. It will enable the seamless integration of diverse block devices into the Android ecosystem. We can envision a future where users can effortlessly control and interact with many devices from their Android devices if we can bring out the convenience and flexibility of NBD integration in Android environments.

Bibliography

- [1] J.-M. D. Goyeneche and E. D. Sousa, “Loadable kernel modules,” *IEEE Software*, vol. 16, no. 1, pp. 65–71, 1999. DOI: 10.1109/52.744571. [Online]. Available: <https://doi.org/10.1109/52.744571>.
- [2] E. Eric, A. Brewer, D. Ritchie, and K. Thompson, “The unix time-sharing system,” *Communications of the ACM*, vol. 17, Apr. 2000. DOI: 10.1145/361011.361061.
- [3] K. Kim, J.-S. Kim, and S.-I. Jung, “Gnbd/via: A network block device over virtual interface architecture on linux,” in *Proceedings 16th International Parallel and Distributed Processing Symposium*, IEEE, 2002. DOI: 10.1109/ipdps.2002.1015476. [Online]. Available: <https://doi.org/10.1109/ipdps.2002.1015476>.
- [4] C. P. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. S. Lam, and M. Rosenblum, “Optimizing the migration of virtual computers,” *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 377–390, Dec. 2002. DOI: 10.1145/844128.844163.
- [5] E. Gal and S. Toledo, “Algorithms and data structures for flash memories,” *ACM Computing Surveys*, vol. 37, no. 2, pp. 138–163, Jun. 2005. DOI: 10.1145/1089733.1089735. [Online]. Available: <https://doi.org/10.1145/1089733.1089735>.
- [6] S. Liang, R. Noronha, and D. K. Panda, “Swapping to remote memory over infiniband: An approach using a high performance network block device,” in *2005 IEEE International Conference on Cluster Computing*, 2005, pp. 1–10. DOI: 10.1109/CLUSTER.2005.347050.
- [7] K. Minghao, K. Y. Chyang, and E. K. Karuppiah, “Performance analysis and optimization of user space versus kernel space network application,” in *2007 5th Student Conference on Research and Development*, IEEE, 2007. DOI: 10.1109/scored.2007.4451372. [Online]. Available: <https://doi.org/10.1109/scored.2007.4451372>.
- [8] H. Raj and K. Schwan, “High performance and scalable i/o virtualization via self-virtualized devices,” ACM Press, 2007. DOI: 10.1145/1272366.1272390.
- [9] W. de Bruijn and H. Bos, “Pipesfs,” *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 55–63, Jul. 2008. DOI: 10.1145/1400097.1400104. [Online]. Available: <https://doi.org/10.1145/1400097.1400104>.
- [10] G. Soundararajan, J. Chen, M. A. Sharaf, and C. Amza, “Dynamic partitioning of the cache hierarchy in shared data centers,” *Proc. VLDB Endow.*, vol. 1, no. 1, pp. 635–646, Aug. 2008, ISSN: 2150-8097. DOI: 10.14778/1453856.1453926. [Online]. Available: <https://doi.org/10.14778/1453856.1453926>.
- [11] T. Hirofuchi, H. Ogawa, H. Nakada, S. Itoh, and S. Sekiguchi, “A live storage migration mechanism over wan for relocatable virtual machine services on

- clouds,” in *2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2009, pp. 460–465. DOI: 10.1109/CCGRID.2009.44.
- [12] F. Maker and Y.-H. Chan, “A survey on android vs. linux,” *University of California*, pp. 1–10, 2009.
- [13] G. Soundararajan, D. Lupei, S. Ghanbari, A. D. Popescu, J. Chen, and C. Amza, “Dynamic resource allocation for database servers running on virtual storage,” in *Fast*, vol. 9, 2009, pp. 71–84.
- [14] X. Wu, W. Wang, B. Lin, and K. Miao, “Composable io: A novel resource sharing platform in personal clouds,” Springer Berlin Heidelberg, 2009, pp. 232–242. DOI: 10.1007/978-3-642-10665-1_21.
- [15] A. Shabtai, Y. Fledel, and Y. Elovici, “Securing android-powered mobile devices using selinux,” *IEEE Security & Privacy Magazine*, vol. 8, no. 3, pp. 36–44, May 2010. DOI: 10.1109/msp.2009.144. [Online]. Available: <https://doi.org/10.1109/msp.2009.144>.
- [16] W. Wang, Y. Zhang, X. Liu, M. Zhang, and Z. Wang, “Hardware assisted resource sharing platform for personal cloud,” IEEE, 2010. DOI: 10.1109/iccet.2010.5486236. [Online]. Available: <http://dx.doi.org/10.1109/ICCET.2010.5486236>.
- [17] Y. Dong, H. Zhu, J. Peng, *et al.*, “Rfs,” *ACM SIGOPS Operating Systems Review*, vol. 45, no. 1, pp. 101–111, Feb. 2011. DOI: 10.1145/1945023.1945036. [Online]. Available: <https://doi.org/10.1145/1945023.1945036>.
- [18] C. E. Palazzi and M. Ferrarese, “Ftp4android: A local/remote file manager for google android platform,” in *2011 IEEE Consumer Communications and Networking Conference (CCNC)*, 2011, pp. 373–377. DOI: 10.1109/CCNC.2011.5766494.
- [19] H. Kim, N. Agrawal, and C. Ungureanu, “Revisiting storage for smartphones,” *ACM Transactions on Storage*, vol. 8, no. 4, pp. 1–25, Nov. 2012. DOI: 10.1145/2385603.2385607. [Online]. Available: <https://doi.org/10.1145/2385603.2385607>.
- [20] H. T. Al-Rayes, “Studying main differences between android & linux operating systems,” *International Journal of Electrical & Computer Sciences IJECS-IJENS*, vol. 12, no. 05, pp. 46–49, 2012.
- [21] A. Stanik, M. Hovestadt, and O. Kao, “Hardware as a service (haas): Physical and virtual hardware on demand,” IEEE, Dec. 2012. DOI: 10.1109/cloudcom.2012.6427579.
- [22] P. González-Férez and A. Bilas, “Tyche: An efficient ethernet-based protocol for converged networked storage,” in *2014 30th Symposium on Mass Storage Systems and Technologies (MSST)*, 2014, pp. 1–11. DOI: 10.1109/MSST.2014.6855540.
- [23] Q. Han, S. Liang, and H. Zhang, “Mobile cloud sensing, big data, and 5g networks make an intelligent and smart world,” *IEEE Network*, vol. 29, no. 2, pp. 40–45, Mar. 2015. DOI: 10.1109/mnet.2015.7064901. [Online]. Available: <https://doi.org/10.1109/mnet.2015.7064901>.
- [24] D. Hein, J. Winter, and A. Fitzek, “Secure block device – secure, flexible, and efficient data storage for arm trustzone systems,” in *2015 IEEE Trustcom/Big-DataSE/ISPA*, vol. 1, 2015, pp. 222–229. DOI: 10.1109/Trustcom.2015.378.

- [25] X. Zhao, Y. Zhang, and B. Su, “Multitask oriented gpu resource sharing and virtualization in cloud environment,” Springer International Publishing, 2015, pp. 509–524. DOI: 10.1007/978-3-319-27122-4_35.
- [26] J. Jung and Y. Won, “Nvramdisk: A transactional block device driver for non-volatile ram,” *IEEE Transactions on Computers*, vol. 65, no. 2, pp. 589–600, Feb. 2016. DOI: 10.1109/tc.2015.2428708. [Online]. Available: <https://doi.org/10.1109/tc.2015.2428708>.
- [27] Z. Li, X. Wang, N. Huang, *et al.*, “An empirical analysis of a large-scale mobile cloud storage service,” ser. IMC ’16, Santa Monica, California, USA: Association for Computing Machinery, 2016, pp. 287–301, ISBN: 9781450345262. DOI: 10.1145/2987443.2987465. [Online]. Available: <https://doi.org/10.1145/2987443.2987465>.
- [28] A. Tzanakaki, M. P. Anastasopoulos, and D. Simeonidou, “Converged optical, wireless, and data center network infrastructures for 5g services,” *Journal of Optical Communications and Networking*, vol. 11, no. 2, A111, Nov. 2018. DOI: 10.1364/jocn.11.00a111. [Online]. Available: <https://doi.org/10.1364/jocn.11.00a111>.
- [29] H. Muhammad, L. C. V. Real, and M. Homer, “Taxonomy of package management in programming languages and operating systems,” in *Proceedings of the 10th Workshop on Programming Languages and Operating Systems*, ser. PLOS’19, Huntsville, ON, Canada: Association for Computing Machinery, 2019, pp. 60–66, ISBN: 9781450370172. DOI: 10.1145/3365137.3365402. [Online]. Available: <https://doi.org/10.1145/3365137.3365402>.
- [30] Q. He, Z. Yu, G. Bian, W. Zhang, K. Liu, and Z. Li, “Research on key technologies of nbd storage service system based on load classification,” *AIP Advances*, vol. 11, no. 12, p. 125 124, Dec. 2021. DOI: 10.1063/5.0071929. [Online]. Available: <https://doi.org/10.1063/5.0071929>.
- [31] D. McKee, Y. Giannaris, C. Ortega, *et al.*, “Preventing kernel hacks with hakcs,” *ndss-symposium*, Jan. 2022. DOI: 10.14722/ndss.2022.24026. [Online]. Available: <https://doi.org/10.14722/ndss.2022.24026>.
- [32] Wikipedia, *Unix philosophy — Wikipedia, the free encyclopedia*, <http://en.wikipedia.org/w/index.php?title=Unix%20philosophy>, [Online; accessed 17-September-2023], 2023.
- [33] *Android open source project*, <https://source.android.com/docs/security/features/encryption>, [Accessed 19-09-2023].
- [34] V. Bhangе, N. S. Negi, and D. Mistry, “Log into android mobile to fetch the device oriented information using remote access via a cloud,”
- [35] *Buy iphone 15 and iphone 15 plus*, <https://www.apple.com/shop/buy-iphone/iphone-15>, [Accessed 20-09-2023].
- [36] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, “Grid information services for distributed resource sharing,” IEEE Computer Society. DOI: 10.1109/hpdc.2001.945188.
- [37] O. R. R. A. C. L. N. B. Device, *Implementing oracle rac using nbd*. [Online]. Available: <https://www.fi.muni.cz/~kripac/orac-nbd/>.
- [38] B. Tierney, W. Johnston, B. Crowley, G. Hoo, C. Brooks, and D. Gunter, “The netlogger methodology for high performance distributed systems performance analysis,” IEEE Computer Society. DOI: 10.1109/hpdc.1998.709980.

- [39] *Udev(7) - linux manual page*. [Online]. Available: <https://man7.org/linux/man-pages/man7/udev.7.html>.
- [40] *Waydroid - waydroid*. [Online]. Available: <https://docs.waydro.id/>.