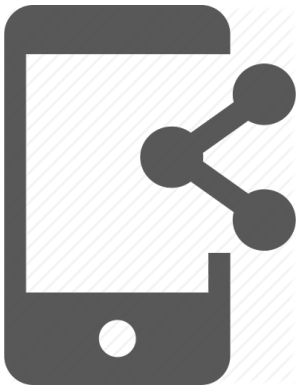
TÉCNICO LISBOA



Termite: Emulation Test-bed for Encounter Networks

Filipe Alexandre Lourenço Teixeira

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Prof. Paulo Jorge Pires Ferreira

Examination Committee

Chairperson: Prof. Daniel Jorge Viegas Gonçalves
Supervisor: Prof. Paulo Jorge Pires Ferreira
Member of the Committee: Prof. Renato Jorge Caleira Nunes

November 2016

To God, my love and my family...

Acknowledgments

First of all, I would like to thank my supervisor, Professor Paulo Ferreira for the support, guidance and motivation given throughout this thesis.

I want to thank Teacher Fátima Lourenço for the support she gave in correcting this document text.

I would like to thank my girlfriend for helping me go through all the challenges, disappointments and achievements. Without you I would never been able to do this, or anything else.

I also want to thank my parents, for their love, patience and resilience. Not only during this thesis process, but also in the time spent at Instituto Superior Técnico(IST).

I would like to thank everyone at INESC-ID that helped me with this project, specially to Professor Nuno Santos and Rodrigo Bruno for the technical support they gave me when I most needed.

Finally, I want to acknowledge the effort made by all my friends and colleagues. You did not only made the journey easier, but you made it possible.

Thank you all.

Resumo

A presença de dispositivos móveis no dia a dia tem crescido num ritmo acelerado. Dispositivos como smartphones ou tablets, têm vindo a evoluir cada vez mais, tanto a nível de performance, como capacidade energética e até mesmo em comunicações. Este último avanço permite às aplicações móveis tirar partido de encontros entre utilizadores, dentro do mesmo espaço físico. Mesmo assim a capacidade de teste e depuramento das ferramentas disponíveis não corresponde de forma suficiente às necessidades de aplicações que utilizam redes baseadas em encontros.

Por essa razão apresentamos o Termite. O Termite é uma ferramenta de emulação e teste de aplicações que utilizem redes baseadas em encontros. Motivado pela necessidade de suportar o teste e depuração de aplicações em redes emuladas, o Termite permite ao utilizador criar o modelo da sua própria rede. O Termite também disponibiliza pontos de paragem e execução passo a passo, facilitando o trabalho do programador. Através da utilização de uma variante única de Petri Nets, é possível modelar dinamicamente a topologia da rede e traduzir a interação dos utilizadores com dispositivos virtuais.

A arquitectura do Termite foi desenvolvida de forma a ser eficiente a gerir múltiplas infraestruturas para auxiliar programadores. Este desenho permite ao Termite suportar plataformas móveis heterogéneas. O protótipo desenvolvido está implementado em Android, num dispositivo virtual, que opera sobre a tecnologia Wi-Fi Direct e que pode ser executado numa infraestrutura do tipo núvem.

Descrevemos ainda uma prova de conceito que permite avaliar a performance do Termite, e que o categoriza em termos de usabilidade em casos de utilização real.

Palavras-chave: Termite, dispositivos móveis, redes baseadas encontros, redes ad-hoc, redes móveis ad-hoc, Android, WiFi-Direct

Abstract

The mobile device presence in everyday life is growing faster and faster. Devices like smartphones and tablets keep growing in performance, battery life and communication. The communication improvement allows mobile applications to take advantage of encounters between co-located users. Nevertheless, the capability of testing and debugging tools available is not enough to deal with the demand of encounter network applications.

So we present Termite. Termite is an emulation test-bed for encounter network applications. Based on the necessity to support application testing and debugging in encounter networks emulations, Termite allow developers to model their own encounter network. By using a unique Petri Net variant it is possible to model topology in a dynamic way and translate user interactions with virtual devices. Through its breakpoints and step-by-step execution, Termite helps in the developers job.

Termite's architecture was designed to be efficient in managing multiple hosting infrastructures to help developers. Termite design also allows to support heterogeneous mobile platforms. The developed prototype implementation is done in Android, in virtual devices through Wi-Fi Direct networks. These networks can run on top of local or cloud infrastructure.

The evaluation made can not only prove that Termite is expressive and performs well, but also that the system have good usability and corresponds to the expectations under real-case scenarios.

Keywords: Termite, Test-bed, Mobile, Mobile Devices, Encounter Networks, Ad-hoc Networks, Opportunistic Networks, peer-to-peer Networks, Android, Wi-Fi Direct

Contents

Acknowledgments v
Resumo vii
Abstract ix
List of Tables xiii
List of Figures xv
Nomenclature 1
Glossary 1

1 Introduction **1**
1.1 Motivation 1
1.2 Goals 2
1.3 Contributions 4
1.4 Document Structure 4

2 Related Work **5**
2.1 Encounter Networks 5
2.2 Network Emulation 7
2.3 Mobile Frameworks 10
2.4 Automated Test Frameworks 13
2.5 Termite Background 14
2.6 Summary 15

3 Architecture **17**
3.1 Background 17
3.1.1 Devel Session Operation 19
3.2 Solution Architecture 19
3.2.1 Client-Controller Interaction 20
3.2.2 Controller Event Management 21
3.2.3 Termite Master Process - EDI Interaction 22
3.2.4 Message Forwarding 23
3.3 Termite Use Case 25
3.4 Summary 26

4 Implementation **27**
4.1 Termite Overview . . . 27
4.1.1 Android and Wi-Fi Direct . . . 27
4.1.2 Android Framework . . . 28
4.1.3 Platform Connectors . . . 29
4.2 Background . . . 30
4.2.1 Design . . . 30
4.2.2 Termite Library . . . 31
4.3 Termite Prototype . . . 31
4.3.1 Design . . . 32
4.3.2 API Integration and Update . . . 33
4.3.3 Flow Control - Android Broadcast . . . 33
4.3.4 Emulation Controller . . . 34
4.3.5 Socket Redirection . . . 35
4.3.6 Connector Update . . . 37
4.4 Summary . . . 37

5 Evaluation **39**
5.1 Network Creation and Deployment . . . 39
5.2 Functionality and Usability . . . 40
5.3 Emulated Disk Image Performance . . . 43
5.4 Summary . . . 44

6 Conclusions **47**
6.1 Future Work . . . 48

Bibliography **49**

A Test Scripts and Files **55**

B Performance Test Results **59**

List of Tables

2.1 Summary table for systems presented. 16

5.1 Workload execution times in seconds 40
5.2 Image send time in milliseconds 44

B.1 Workload execution times in seconds 60
B.2 Image send time in milliseconds 61

List of Figures

2.1 Wireless Mesh Network illustration[13]. 6
2.2 Termite layers focusing network emulation. 7
2.3 Termite layers focusing on the mobile integration. 11
2.4 Termite layers focusing application testing. 13

3.1 Termite front-end and back-end interaction[53]. 18
3.2 Actors involved in T-Net emulation[53]. 18
3.3 Interaction between the Master Process and the Controller inside the Emulated Disk Image 20
3.4 Creation of 2 nodes network in Termite 22
3.5 Connection between EDIs 23
3.6 Connection between EDIs 24
3.7 Group Owner-Nodes Connection 25
3.8 Broadcast Protocol example 25
3.9 Usecase Ouput 26

4.1 Android Layers. 28
4.2 Android WifiP2pManager Class Implementation example [60]. 29
4.3 Termite 1.0 Implementation Design 30
4.4 Termite Implementation Design 32
4.5 Termite API Integration 33
4.6 Screenshot of Settings Menu 35
4.7 Socket Redirection in Termite 1.0 36
4.8 Socket Redirection in Termite 36

5.1 Code snipe of Android WifiP2p and Termite 1.0 Initialization. 41
5.2 Code snipe of Android WifiP2p and Termite 1.0 Event Registration. 42
5.3 Code snipe of Android BroadcastReceiver. 43
5.4 Code snipe of Termite 1.0 BroadcastReceiver. 44
5.5 Code snipe of Android socket usage. 45
5.6 Code snipe of Termite 1.0 socket usage. 46

A.1 Termite Script code snipe for the Workload 50 Test. 55

A.2 Termite Script code snipe for the Workload 100 Test. 56
A.3 Termite Script code snipe for the Workload 250 Test. 56
A.4 5MB Image used for Termite Performance Test. 56
A.5 9,7MB Image used for Termite Performance Test. 57

Chapter 1

Introduction

In the last years, mobile devices became the primary computational platform for users[1]. With officially more devices than people in the world[2], mobile devices have become natural in the everyday life. Although there are countless usages of a mobile device, they are mainly used to store and share information.

The exponential increase of devices, brought new communication paradigms that did not exist before. Although performance has increased, device communication capabilities did not follow this improvement. Different technologies, such as Bluetooth or Wi-Fi, have been improved to support the communication demands among mobile devices. Nevertheless, the technology development did not fulfil the peer-to-peer communication demands.

Almost all datasharing applications demand for some form of connection. The main way of communication is through internet, using mobile data, like GSM, or Wi-Fi. For example when two users share a file in the same physical environment. The most commonly used method for this file sharing establishes a connection to a central server or some kind of redirection to exchange data. This example shows the lack of peer-to-peer and peer group solutions. To solve this problem a new paradigm has emerged, the Encounter Networks paradigm[3].

The term Encounter Networks derives from the ad-hoc networks but in a much more specific point of view: Encounter Networks refer to the special case of ad-hoc networks that target co-located moving mobile devices and involve social interactions.

1.1 Motivation

Consider the scenario where a mobile application developer, Joe, wants to create a new kind of application. Joe wants to build an application that shares files between two mobile devices that are in the same

physical space. Scalability is one of the application requirements that Joe should take into account. To test the application scalability, Joe had to use several devices. With the current technology Joe would have to gather dozens of mobile devices to perform accurate tests. Once Joe is an entrepreneur, his limited resources does not allow him to test the application scalability. Reliability of the tests is also a problem. Joe is unable to reproduce exactly the individual user displacement for an accurate application testing. Therefore, Joe will publish his application without being fully tested or will give up this kind of developing as the result of available test solutions.

Another approach is the use of a cloud server. This server would hold all the application information and create the connections among mobile devices. This solution has advantages and disadvantages concerning other solutions. Having an internet server allows the user to reach other clients that are not within his range. For example, lets consider Joe wants to share a file with an user that is not nearby. Joe can interact with any user who has an internet connection. However, if the user is within Joe network, this solution would take longer, and consume mobile bandwidth. Another problem relates to the mobile paradigm. The user can have limited bandwidth, not allowing him to share a file with a friend who is just one step away. This case shows that Cloud Services sometimes are not the best solution.

These scenarios portray the motivation for the development of testing tools. These tools can help programmers to acquire a better understanding of the technology while improving several aspects of the mobile applications development.

There was a lot of work creating test-beds and developer tools. The most common mobile platforms, like Android and iOS, have their own development tool kits[4, 5]. These tool kits already include virtual device emulators and frameworks for automated UI testing. However, these systems do not provide the necessary support in order to develop and test applications based on the Encounter Network paradigm. These tools do not implement multi-node emulation which is required to test encounter network scenarios. There are other alternatives, but most of them have one of two drawbacks: either do not support correct network emulation, or they do not have the correct UI automated support for testing.

1.2 Goals

The main goal of this project is to develop Termite, a system that can emulate the Encounter Networks paradigm and support the development and test of mobile applications.

Termite is a tool built to increase the speed of the development cycle in order to help developers creating Encounter Based Networks Applications. Termite creates a test-bed that allows developers testing and debugging their applications. Concerning the typical development cycle, the programmer starts by coding the application. The following phases are assisted by Termite. Termite allows the developers to build their own Encounter Based Network, emulating the expected behaviour of that real world network.

This network is created considering emulated nodes that represent mobile devices. After the network is created, Termite helps developers to deploy their applications and run automated tests producing output information. The information gathered from this tests is crucial for developers to understand the behaviour of their applications. Termite also allows the programmer to run the system in a debug mode where he can use tools like: step-by-step execution, breakpoints while inspecting the execution state of specific nodes.

Termite has two components. One is the Termite front-end, where the developer can command emulators, the emulated network, and the test and debug tools. The other is the image used in emulators. Termite should be able to test and debug real world applications without requiring specific changes in the applications source code. Termite provides the image that is used in virtual devices emulation. The image is expanded with Termite extensions, whose importance ensure the system is transparent to the application.

Thus, Termite must fulfil the following requirements:

Flexibility. Termite must be flexible to support a wide range of devices. Emulated devices must be supported by Termite. Device emulation can exist in the same physical machine or in a cloud server. Physical devices must be supported too. Encounter based Networks should be created by using either one or both types of devices, i.e. a mix of real and emulated.

Performance. Termite must have a good performance. The system performance must be very close to the stock system in physical devices. Emulated devices performance must also fulfil this requirement taking into account emulation limitations.

Robustness. The system must be able to handle expectable user interactions as well as unexpected ones, tolerating any devices or connection failures in an efficient way.

Scalability. The emulation protocol must ensure the system scalability. This means the system must be able to handle a growing amount of devices and be able to accommodate that growth.

Usability. Helps the developer to start testing and debugging his application quickly. Therefore, the system must be easy and fast assembled with no changes in the application code. The system must have a clear and easy user interface.

Portability. Offering multiple solutions for the developer, the system must be compatible with multiple types of platforms and devices. To do this the system must rely on Wi-Fi Direct technology to create Encounter based Networks.

To create a system that fulfils all the above mentioned requirements some challenges have to be fulfilled:

- Integrate Termite in the mobile device system without harming other functionalities;
- Correctly simulate the real word device interaction, and all possible use scenarios;

- Develop an intuitive tool for emulation control and movement simulation;
- Ensure that all emulator instances are coordinated;
- Replicate the interaction between physical and emulated devices, specially the offloaded images.
- Provide test and debugging functionalities;

1.3 Contributions

This work provides the following contributions:

- architectural design of Termite test bed as well as the interactions, protocols and procedures;
- architectural design of the Emulation Image used for the mobile devices in the Termite system;
- implementation of new connector between the Termite and the Mobile Emulated devices;
- implementation of the Mobile Emulation Image augmentation in an Android prototype;
- evaluation of the emulation effects in the Encounter Based Network environment;
- evaluation of Termite performance impact comparing to real world scenarios.

1.4 Document Structure

This document is organized as follows. In Chapter 2, we describe different kinds of systems related to Termite goals. Then, Chapter 3 presents our solution, describing Termite background and the architectural development. Chapter 4 describes the implementation of Termite prototype, and Section 5 presents the evaluation of the system. Finally, Chapter 6 presents conclusions and suggests future developments.

Chapter 2

Related Work

This Chapter is organized in five main parts: *(i)* Encounter Networks paradigm, *(ii)* Network Emulation tools and frameworks, *(iii)* Mobile Middleware and Frameworks, *(iv)* Automated Test Frameworks and Applications, and *(v)* Termite Background.

On the first section, the ”Encounter Networks” paradigm is discussed and compared with other known paradigms. We address differences between the current ad-hoc networks definitions and the Encounter Networks description. Section 2.2 focus on network emulation methods, looking at different tools and frameworks that present solutions to network emulation. Section 2.3 targets development frameworks focused on the mobile paradigm. This section describes systems that intend to give an answer to mobile problems, specially in the communication area. Later on, Section 2.4 approaches frameworks for test automation, with an emphasis on UI automation support. Section 2.5 is centred in Termite background and its pitfalls. Finally, Section 2.6, summarizes the most important aspects and evaluates the current solutions.

2.1 Encounter Networks

Encounter Networks address the case of ad-hoc networks that target co-located mobile devices and involve social interactions. However Encounter Networks can be compared to other network definitions such as Mobile Ad-Hoc Networks. Jo et al.[6] compare several networks, relating wireless networks with mobile ad-hoc networks. In this section we describe several network definitions, comparing them with the Encounter Network paradigm.

Mobile Ad-Hoc Networks or simply MANETs[7] are a well researched area but still with some pitfalls. One of them is the secure MANETs bootstrapping. Xu et al.[8] focus on MANETs secure bootstrapping in scenarios where the network users do not share trust relationships prior to the network deployment. Another pitfall is the packet forwarding inside the network. Zouridaki et al.[9] offer a solution with a robust cooperative trust establishment scheme to improve the reliability of packet delivery. This problem is not

so important in the Encounter Network paradigm. MANETs have several uses. One of these uses is presented by Papandrea[10]. In her paper we can find a system which allows the identification of mobile nodes based on a derivative of MANETs, opportunistic mobile networks.

Opportunistic Networks are described as mobile ad-hoc networks[7] focused on the continuous update of highly dynamic topologies[11, 12]. While Opportunistic Networks are focused on dynamic topologies, Encounter Networks also consider social interactions and user movement. In the article[11] focus on developing a Wi-Fi Direct framework to support the creation of Opportunistic Networks. Although the project improves the current Wi-Fi Direct protocol, it still presents the same development and testing problems found on the stock WiFi Direct framework has. The framework was developed for physical devices use only. Therefore, the testing nodes are limited to the number of physical devices available. As Opportunistic Networks are not taken into account the user location, the framework does not support user displacement testing. It is not possible to simulate users mobility in an automated test.

WiFi-Opp[12] is very similar to the previous system. The main difference is that in WiFi-Opp, Opportunistic Networks are directly created through WiFi technology without using the WiFi Direct API. Although this change presents some design and performance improvements, the same drawbacks still occur. There is no support for emulated devices or user displacement simulation.

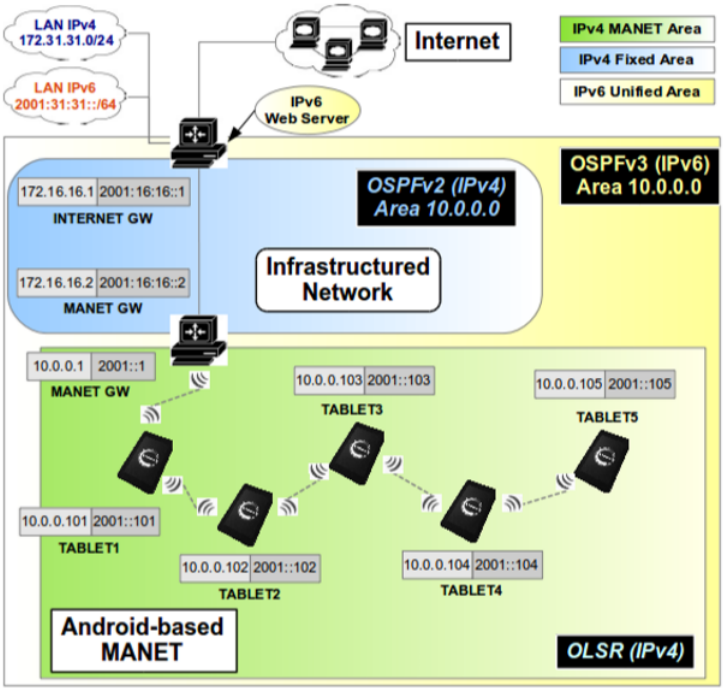


Figure 2.1: Wireless Mesh Network illustration[13].

Another type of ad-hoc mobile networks is the Wireless Mesh Networks (WMNs). WMNs are described as MANETs, where the nodes are not only user devices. In this paradigm, the infrastructure is considered part of the network. This network model includes, for example, printers or gateways. Fig.2.1

illustrates the concept of WMNs. Pratesi et al.[13] propose a test-bed for the development of Android applications that takes advantage of these networks. Besides being more complex than the previous examples, it still has the same drawbacks. This system presents no solution for scalability tests other then the use of physical devices. It also does not present test automation or user displacement emulation.

The closest description to Encounter Based Networks is made by Korhonen[14] and first quoted by Kurhinen et al.[15]. Mobile Encounter Network or simply MEN describes the encounter between devices communicating while moving that allows ad-hoc -type data transfer to be performed. In [3] we can see some applications of mobile encounter networks. Although MENs and Encounter Based Networks are very similar, the support offered by the references is very conceptual. Thus, a concrete solution to develop applications based on this paradigm fails.

2.2 Network Emulation

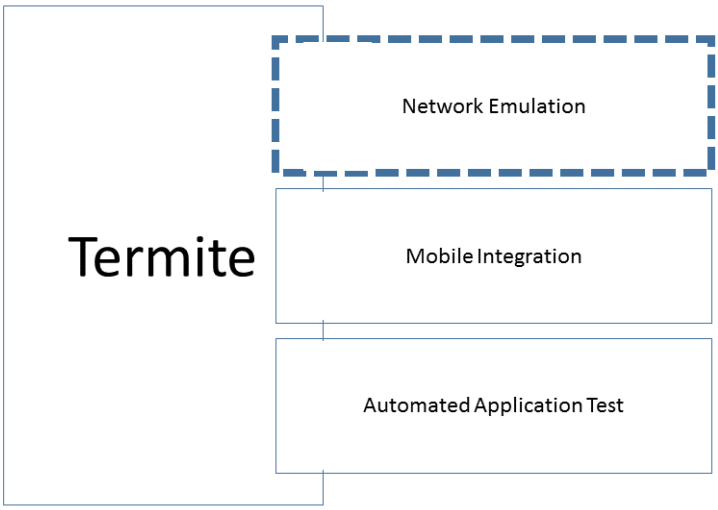


Figure 2.2: Termite layers focusing network emulation.

To emulate an Encounter Network, it is necessary to emulate the background infrastructure that supports the mobile interaction and communication. There are several available network emulation solutions to do so. These solutions are based on low-level network protocols and do not provide the necessary test automation required by this project. The systems presented in this section focus on network emulation. Network emulation is used to test real applications over a virtual network, and focus the network layer of Termite, as seen in Fig.2.2.

Encounter Network based applications development add to the development cycle the challenge of device mobility. Establishing mobile ad-hoc networks between devices brings up problems like configu-

ration, routing and name resolution[16]. Some of these problems are solved by WiFi-Direct but other pitfalls still remain. The differences between routing protocols have impact in the network performance[17]. Mobility cases[18] can also interfere with application execution. Therefore, it is important to test and debug Encounter Based Networks applications in every possible way. In this section, we present some solutions that focus on network emulation flexibility.

Some of these solutions only provide network simulation, like NS[19]. Based on this network simulation protocol Kevin Fall from University of California developed a Network Emulation framework[20]. Despite the improvement of this tool, it still does not match the requirements proposed on Chapter 1. The improved NS only supports network emulation and do not fulfil the automation methods for correct development and test of Encounter Based Network applications. NS[19] and the Network Emulation framework[20] are very important tools once almost all systems presented in this section are related, based or even created upon the NS system.

EmuNET[21] is an extendable network emulator. This emulator creates a re-routing table allowing the packets to be delivered as in the real network. Despite being simple, the user would have to configure manually the amount of nodes present in the network which would limit the scalability of the system. EmuNET also provides several network features to produce real life events in the network. Some examples are Network Queues, Traffic Shaping, Delay and Jitter, Bit Rate Limiting and Packet Loss. This tool affects the network through packet capturing which can be a problem in a distributed system. Another pitfall of this tools is the fact that it does not support changes in the network during runtime. EmuNET also does not support any kind of specific Mobile focusing. Due the lack of automate tools for application testing, the developer would need to have more than one tool to test his applications. This would not only create problems in scalability but it would also be very difficult to coordinate and replicate network conditions with application testing scenarios.

There are some emulators built with distributed properties, this means that the emulated network can be built over an internet connection. That is the case of NetWire[22]. NetWire is a network emulator built to teach and understand networks. It is important to mention it, because it has also another important goal, to provide a distributed architecture. This tool allows the emulation of networks in a distributed layout. NetWire emulates the network at the physical and MAC layer of the ISO/OSI network model which allows it to be very efficient and consistent. It can emulate a wide range of network topologies, basic physical characteristics of the communication channels and allows parameters to be changed in runtime.It provides some improvements in the Network Emulation area, however it still has some pitfalls. NetWire is a client/server architecture which can generate some problems in a Encounter Network emulation. Most of the specified problems for EmuNET can also be found on this tool. The lack of mobile support brings some problems to the developer referring to automated application testing.

NIST Net[23] is a linux-based network emulation tool. It was built to provide testing and experimen-

tation with network code through emulation. This tool is very similar to EmuNET[21], but instead of working as a packet capturer and simply re-routing packets, NIST Net works as a ”network in a box”. This system works as a specialized router which emulates the behaviour of an entire network. As a result of this feature, the network is also static and does not allow runtime modifications. NIST Net presents the same flaws as EmuNET, lack of mobile support, no distributed solutions and it is only focused on wired network problems.

Previous systems focused on problems like packet loss and delay however, network emulation based on wireless emulation considers it as the consequences of user mobility. Weingartner et al.[24] presents a new network emulation architecture that target wireless communication software. Through the development of a custom device driver, this system enables arbitrary and unmodified wireless communication. Although improving some of the problems presented in systems like EmuNET[21] and NIST Net[23], this system shares many of the previous systems issues. Reducing the developer configuration effort is a consequence of the focus on wireless networks. However, the lack of application testing support makes testing a difficult task for the programmer. Not supporting emulation over internet connection diminishes testing scalability.

LabVIEW[25] is a test-bed based on off-the-shelf components created to mobile sensor networks research. The most interesting feature of this test-bed is the emulation of the physical interactions between nodes. This movement emulation is called virtual motion. The algorithm presented by LabVIEW allows a dynamic topology of the network. Mobile sensors are the focus of LabVIEW work and not Encounter Networks however, there are still contributions for the user displacement emulation proposed project. Encounter Networks not only represent the mobile sensor networks challenges but much more, for example the UI automation. Therefore, LabVIEW can not be a solution for the requirements presented for Termite.

Another network testing tool, indeed a more developed one, offers more features other than the simple network emulation. One example is WindTunnel[26]. WindTunnel is a tool for network impact testing of mobile applications: an innovative feature is the capacity of simulating user movement and bandwidth problems. This is one of the key Termite project requirements. It is essential to simulate user movement when using emulated devices. However, WindTunnel does not support the capability to use emulated devices and still has the testing problems described for the network emulation tools.

EMWIN[27] is a mobile wireless network emulator. This system allows emulating multiple mobile hosts in a single emulator node. With this feature the system presents a significant improvement in scalability. EMWIN also provides the ability to emulate the mobility of a mobile host in a wired network, what is very important in the Encounter Based Network paradigm. Although presenting several improvements, the system has the same pitfalls expected in a network emulator. The lack of application targeting present a problem to a developer focused in Encounter Based Network applications. Not having appli-

cation testing tools does not allow the system to fulfil the initial requirements presented in this document.

A very similar system to EMWIN is m-ORBIT[28]. m-ORBIT presents improvements in the emulation of mobile networks. Through Spatial Switching on a Wireless Grid, m-ORBIT can emulate a large-scale mobile network with a much more accurate mobility displacement emulation. This improvement creates more stability in the mobile network emulation. However this tool has the same pitfalls as the ones presented in EMWIN.

QOMB[29] is a test-bed designed and implemented for the evaluation of wireless network systems, protocols and applications. This system uses the emulation tools of QOMET[30] and the StarBED[31] infrastructure. StarBED is a large-scale network test-bed developed at the National Institute of Information and Communcations Technology in Japan. This network counts with a cluster of 1000 standard PCs. QOMB can successfully emulate a mobile ad-hoc network and provide the tools to emulate a mobile network behaviour. The QOMB evaluation[32] concludes that the QOMB system is very close to the results of a real world scenario. With the extended focus on mobile networks, QOMB provides less pitfalls than the previous systems. QOMB offers better integration and configuration but still presents some of the expected network emulators flaws. QOMB does not support mobile application development because it lacks the appropriated automated testing tools. However, by using a distributed system, QOMB does not present scalability problems, but it can only be used on the StarBED infrastructure. The accessibility of the infrastructure can limit developers options.

It is notorious that network emulators do not have the necessary features to support the Encounter based Networks paradigm. Most of the existing emulators do not properly support test automation of Encounter based Network applications.

2.3 Mobile Frameworks

There has been an increasing focus on MANETs with the development of multiple frameworks and middleware focusing on this paradigm. The major problem of these systems is the specific usage. While focusing in a specific problem, these systems do not provide tools to test and debug of encounter based applications. These frameworks and middleware provide services beyond those available in the Operating System. Therefore, they are important to the integration of Termite in the mobile environment. Fig.2.3 shows the layer where mobile frameworks fit in Termite.

An example is the JiST/MobNet system[33]. This tool presents a new approach to provide simulation, emulation, and real-world experiments in mobile ad-hoc networks. This system focuses on the MANET paradigm by targeting the network testing. Presenting a tool focused on network protocol and interaction testing, the test-bed presented by JiST it does not in fulfil the requirements we described in Chapter 1.

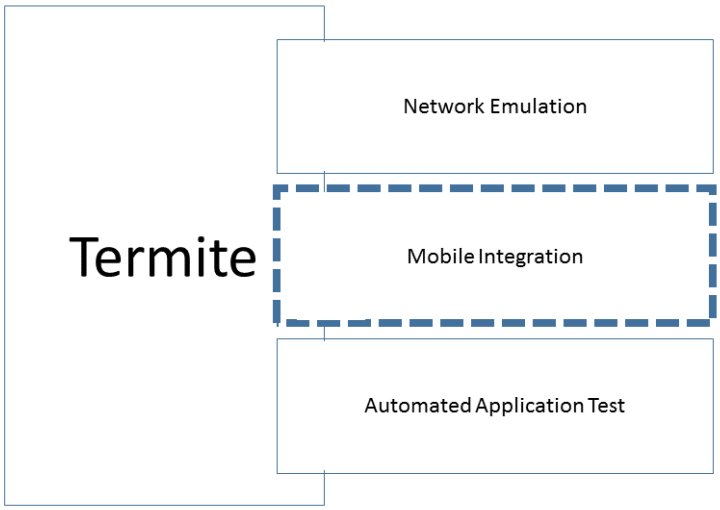


Figure 2.3: Termite layers focusing on the mobile integration.

The JiST test-bed does not provide automated test support which is related with the network emulators described in the previous section.

Another system very close to JiST in what concerns its pitfalls, is the MMAN[34]. MMAN is a monitor for mobile ad hoc networks. This monitor is responsible for monitoring mobile ad hoc networks and provide value network management, security assessment, and traffic analysis information. As stated before this kind of systems are far from the requirements presented for Termite.

Spontaneousware[35] is a framework used to facilitate the development of middleware systems for MANETs. This framework serves as a pre-implementation of the middleware that is developer customizable. Although the framework is useful for MANET middleware development, it does not provide application testing capabilities. The lack of network emulation and application automation are the major pitfalls of Spontaneousware when compared to our requirements.

Other important middleware to mention is ATMOS[36]. ATMOS is a middleware for transparent mobile ad-hoc networking systems. Through a provided virtual link interface, ATMOS can easily implement protocols for MANETs. However ATMOS does not fit in Encounter Based Network paradigm. Although ATMOS provides network simulation capabilities, it does not provide network emulation tools like user movement or packet manipulation. Also, the lack of automated test is an important factor to have into account.

iTrust[37] presents a peer management over Wi-Fi Direct in a MANET. This system was developed for iTrust peer-to-peer information publication. One of the most perceptible flaws is the lack of automation to test applications. There is not also a reference to a test environment using emulated devices, what

causes scalability testing a problem. Other drawbacks are related to the implementation of the Wi-Fi Direct controller. In this peer management the implementation of the tool is overlaying the regular stock implementation. This architecture has a negative impact on the portability of the framework, one of the key requirements established in this project.

When the mobile ad-hoc network paradigm is discussed, the mobile social network is an important subject, an example is MobiClique[38]. MobiClique is a system that, through Bluetooth, creates opportunistic networks. The major improvement of MobiClique when compared to other solutions, is the removal of the central server needed in social networking. Although the developers of MobiClique provided an open API for third-party applications, it is not considered enough to satisfy Encounter Network based applications. When comparing MobiClique with the requirements presented in the previous chapter there are still some difficulties to solve. One is the lack of emulation support. Another one is the lack of user movement emulation, a consequence from the mobile social networking paradigm. In short, the provided API is not enough to test and develop accurately Encounter Network based applications.

One of the most interesting tools of the MANET paradigm is Java Mail NFC (JNFC)[39]. JNFC is an emulation of the NFC protocol in virtual devices. To test this application the developers created Droid-WSN Wireless Sensor Network (WSN). WSN emulates the ad-hoc network created by the Near Field Communication (NFC) technology. This system is interesting because it is one of the few frameworks related to MANET paradigm that focus on emulated devices. This allows the application developer to discard the need of having several physical devices to test his application. However, there are still some flaws: no user displacement emulation, no UI automation support or the use of NFC technology. There are also flaws in the architecture and system implementation. When analysing the results of the framework, the authors consider there is a performance problem within the tool. This problem may be related to the use of email to send the NFC information between emulators which is clearly much slower that a normal socket connection.

Another presented platform is mQual[40]. mQual is a mobile peer-to-peer network framework, implemented as an upgrade to the Wifi-Direct framework. One of the most interesting features of this system is the implementation. Taking advantage of the already implemented Wifi-Direct framework, mQual extends its capabilities to adjust networks. The network adjustment helps to fulfil the network requirements in dynamic environments. With mQual being essentially a performance upgrade to the Wifi-Direct protocol, the same drawbacks described in Chapter 1 still apply. The article does not mention the use of emulators in the system, one of Termite's main requirement. With the lack of requirements this article does not influence the architecture of Termite, but it can help to understand the implementation details under Android WiFi-Direct Framework.

2.4 Automated Test Frameworks

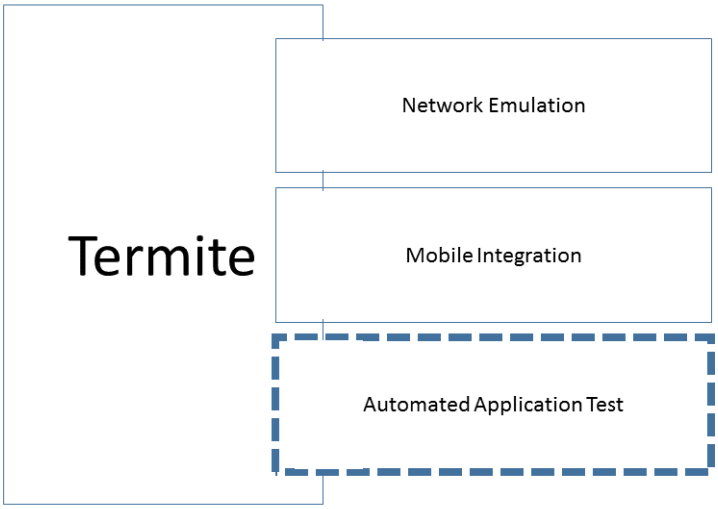


Figure 2.4: Termite layers focusing application testing.

Application test automation has been a point of interest since the advent of application development. Unlike other programmes that can be tested through test battery, applications add the UI factor to the equation. UI test automation tools have presented huge improvements. Haller[41] shows the importance of mobile testing and concludes that test environment stability and test automation are crucial in application development. However, simple application testing has already enough fragilities[42], encounter based application add the distributed factor issue. Termite needs to provide application test support to the developer. Application test support is provided by the application test layer in Termite, as seen in Fig.2.4. This section focuses on the limitation of existing automated test frameworks when applied to the Encounter Networks Paradigm.

MonkeyRunner[43] and Espresso[44] are simple tools which provide APIs to control the execution flow of an application test. However this type of tools are not complex enough to have the emulation capability needed to test encounter network applications. These tools can be integrated in a emulation framework to add the necessary execution flow control to an emulation framework. This is possible, but the complexity in coordinating both tools would be higher than building the entire system from scratch.

PlanetLab[45] and EmuLab[46] are frameworks ready for emulation tests. These frameworks have the capability of aiding the system development, testing and debugging. The main problem, in this solution, is the lack of the appropriate abstraction methods in order to model encounter networks.

Although there are a lot of commercial test application frameworks and tools, there are still improvements in this area. MobileTest[47] is a single target test automation that allows application testing through its

framework. The paper[48] complements MobileTest by improving the framework while paying special attention to scalability. The same pitfalls of the previous systems still apply to MobileTest. The lack of network emulation makes the testing of Encounter Based Networks application harder.

The implementation of the cloud paradigm in application testing has been widely discussed. An example is the conceptual description of a cloud based mobile application testing made by Calpur[49]. The usage of the cloud can greatly increase the scalability of the system. Testdroid[50] implements a remote UI testing for Android. Taking advantage of an online platform, the developer can perform application tests in a wide variety of Android handsets. Although these systems increase the scalability of application testing, the lack of network emulation between the devices create several barriers to the testing of Encounter Based Networks applications.

The most interesting system is the one described in [51]. This is a framework for remote automation, configuration, and monitoring of real-world experiments. The framework allows testing applications in a distributed environment. The system also allows a network simulation between the used devices. This framework is implemented over the CRAWLER[52] cross-layer architecture. Although this system fulfils some of the presented requirements, the focus on encounter based networks application testing is still a pitfall. The framework does not provide the emulation of user movement or network manipulation. For the testing of encounter based networks application it is very important to support the emulation of normal and abnormal user behaviour.

2.5 Termite Background

Termite 1.0[53] is the closest system to the one described in this document. Termite 1.0 is the main inspiration of this project. Although both systems requirements are very similar, Termite 1.0 does not respect some key aspects of Termite. Because of the importance of these aspects, it was necessary to re-evaluate the goals and recreate Termite 1.0 through this project. In this section we compare the goals and requirements of both systems, presenting their similarities and differences, analysing both architecture and implementation and identifying the main reasons why is Termite a necessary upgrade to Termite 1.0.

When comparing the underlying paradigm that originates both systems we can observe that they are close to each other. These systems focus on the Encounter Network paradigm, while their main goal is to support testing and debugging of Encounter Network applications. Both systems are based on the lack of solutions for testing and debugging of applications that target co-located moving personal devices and involving social interactions.

In Termite 1.0, different aspects are targeted. The first one is the notion of Termite specifications or

T-Net specs. These T-Net specs describe the topology evolution of an encounter network. Although not being very specific, the description only targets virtual devices. One of Termite requirements is the portability of the system which ensures that Termite is compatible with virtual and physical devices. Therefore, Termite 1.0 is not ready for the communication between these two different types of devices.

Termite 1.0 expresses three main requirements, which are expressiveness, reproducibility and execution control. These requirements are included in the requirements presented in Chapter 1. Following the same architectural guidelines, Termite expanded the Termite 1.0 with the requirements described in Chapter 1.

Termite 1.0 architecture is very modular. This allows Termite to expand the existing modules to fulfil the new requirements. Even if the design does not focus on the flexibility of the system, the modular ability of T-Nets allows the integration of new components through the platform-specific connectors. These connectors translate the communications of the Termite with the device emulation.

Although Termite 1.0 model, architecture and design are in line with the new requirements, the implementation presented there conflicts with the requirements defined for Termite. Android and Wi-Fi Direct were the technology chosen to support the implementation of Termite 1.0. A problem of Termite 1.0 implementation is the Android-Specific Place Enforcers. To avoid modifying Android, Termite 1.0 implemented its own library for Wi-Fi Direct emulation. The library implemented created a new API. The developer would then use the new API instead of the Android native API. Termite presents a great improvement in terms of usability when compared with Termite 1.0. Instead of using Termite API, developers can now use the Android API. With the possibility of testing applications without changing the source code, Termite provides a better usability for developers.

Once Termite 1.0 is the background of Termite, we can conclude that some of the requirements described in the first chapter can not be fulfilled by Termite 1.0. The two main unfitted requirements are flexibility and usability, both are very important for the user experience. Termite 1.0 does not support the usage of physical devices in the emulation process. It also does not support direct application testing, since the developer needs to modify the application code to support the usage of Termite 1.0. Termite, as described in this document, supports the usage of physical devices in the emulation process and also allows developers to test their applications using the native code.

2.6 Summary

In this chapter, we present the motivation to provide adequate testing and debugging tools for Encounter Based Network applications. The specific case of Encounter Based Network paradigm creates challenges that were not made to the normal application development cycle. Thus creating several difficulties in completing the application development testing and debugging step. These difficulties originate partly

Table 2.1: Summary table for systems presented.

	Network Emulation	Node Scalability	Dynamic Topology	User Displacement	Mobile Support	UI Test Automation	Native Application Code Support
WiFi Opp Net	X						
WiFi-Opp	X		X				
WMN	X						
Network Emulation framework	X						
EmuNET	X						
NetWire	X	X					
NIST Net	X	X					
LabVIEW	X						
WindTunnel	X			X			
EMWIN	X	X					
m-ORBIT	X	X	X	X			
QOMB	X	X	X	X			
JiST/MobNet	X				X		
MMAN					X		
Spontaneousware					X		
ATMOS	X		X		X		
iTrust	X				X		
MobiClique					X		
Java Mail NFC (JNFC)	X				X		
mQual	X		X		X		
MonkeyRunner					X	X	
Espresso					X	X	X
PlanetLab	X				X	X	
Emulab	X				X	X	
MobileTest					X	X	X
Calpur		X			X	X	X
Testdroid		X			X	X	X
CRAWLER2	X				X	X	
Termite	X	X	X	X	X	X	

from the necessity to extend the normal application execution with user displacement and connection problems, e.g., packet loss or increased delay.

Analysing the related work, we become aware that there are a lot of relevant works in application testing, but none of the existing solution fulfil all of our requirements. Network emulation tools offer several key aspects in emulating Encounter Networks or in emulating user movement and network issues. But the network emulation is not enough on its own. Automated testing is also important to the developer. The test system must provide test replication conditions to the developer. However, network emulation tools do not provide any. On the opposite, Automated Test Frameworks do not offer emulation tools to implement a single solution to Encounter Based Networks application testing. In this chapter we also presented Termite 1.0 that is the background of this work. But we also conclude that Termite 1.0 does not fulfil all the requirements. The testing system must be transparent to the application and Termite 1.0 forces the developer to use a specific API, requiring code changes.

In the Table 2.1 there is summary about the important systems described in this section. We compare each system while considering the requirements described by Termite.

In the next chapter we present a solution to test and debug Encounter Based Network applications that fulfil the initial requirements.

Chapter 3

Architecture

This chapter presents the Termite architecture design. This architecture is based on the Termite 1.0[53]. Termite 1.0 creates a base allowing us to approach the requirements described in Chapter 1. Therefore it is crucial to identify the Termite 1.0 architecture shortcomings in order to extend it and fulfil Termite requirements.

First, we present the architecture design and its components based on Termite 1.0. Later on we explain in detail the scope of this work, the changes made on base components and their usage in the system.

3.1 Background

This section is devoted to the description of the Termite 1.0 architectural design. The goal is to explain Termite 1.0 components and their interaction to set a background for the work presented in this thesis.

Termite design is based in three main requirements. These requirements, specified bellow, provide a better understanding of Termite architecture and its purposes. The first requirement is an emulation service based on Termite Nets (T-Nets). T-Nets are responsible for the layout of the emulated network and it is well documented in Termite 1.0[53]. The second requirement specifies that resource management must be efficient. This means that the hosting infrastructure must be shared across multiple nodes. The third requirement, Termite must be general and through the incorporation of platform-specific, must also support the emulation of arbitrary mobile platforms.

Regarding components, Termite supports two front-ends and multiple back-ends (see Fig. 3.1). The front-ends, two software clients, are composed by an Administration Client that is used for example to create Emulator Disk Images or to manage account settings. The other client is used for Devel Sessions, these sessions will be explained later. The front-ends are dynamically configurable to use the back-end

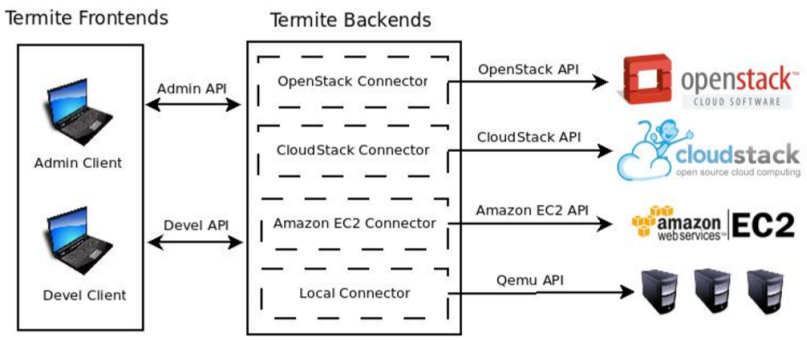


Figure 3.1: Termite front-end and back-end interaction[53].

connectors. These back-end connectors must respect the API used by developer and administrator front-ends. The connectors purpose is to stablish a communication channel to a specific platform. We have a few examples of these connectors in Fig.3.1. The connectors are used to stablish a connection to a cloud platform, like Openstack[54] or Cloudstack[55], or to a local emulation infrastructure.

Termite architecture is articulated to support the emulation of multiple back-end nodes, these nodes could be instantiated in cloud platforms. To support these connections Termite must provide a platform-specific Emulation Disk Image (EDI). This EDI, or just emulated image, must be enhanced with Termite extensions to enforce the T-Net model on all emulator instances booted from that emulated image. Termite improves the Termite 1.0 Emulation Disk Image definition. Instead of using extensions to support Termite, the emulated image must be modelled to support Termite interactions. The modelling solution presented is described in section 2 of this chapter.

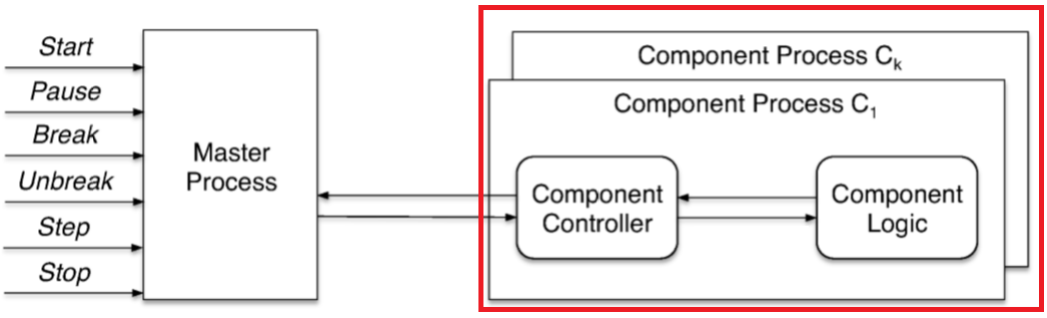


Figure 3.2: Actors involved in T-Net emulation[53].

Fig.3.2 shows the conceptual model of the components, described below, and its interaction. The Master Process is responsible for controlling the emulation and issuing commands to the nodes. The Component Controller works as a communication channel between the Master Process and the Component Logic. The Component Logic emulates the component-specific functionality regarding actions like interface output or network events. In the Fig. 3.2 we can also see highlighted the scope of our work which is the Termite extensions inside the emulated image, or as we referred, the Emulation Disk Image.

3.1.1 Devel Session Operation

Devel Sessions are the main service provided by Termite 1.0 to the users. These sessions allow developers to test and debug their applications. The Devel Client front-end is responsible for the interaction between users and the sessions. The client implements commands to create, set up and control the Devel Session life cycle in coordination with Termite back-end components.

Termite Devel Session Operation is responsible for the following characteristics:

1. **Provide a T-Nets specification:** The developer must provide a specification (spec) of the T-Net. After completing this step, the Devel Client creates a profile of the emulator instances to be created. For each emulator instance, the Devel Client determines the respective emulated image and the EDI parameters, that can be supplied by the developer as well.

2. **Bootstrap emulator instances:** Based on the created profile, the developer must request the instantiation of the emulators on the hosting infrastructure, through the specific connector. After the connector instantiates all the emulators, the client is notified.

3. **Deploy App and T-net spec:** The final step of the set-up phase is deploying the App package and the T-Net spec to the instantiated emulators. After performing this deployment in every instantiated emulator, the emulation phase can be started.

3.2 Solution Architecture

This section describes the new Termite architectural design.

The Emulated Disk Image is the solution to fulfil the Termite requirements and solve Termite 1.0 issues. As explained in section 1, on Termite 1.0, the emulated image is enforced with extensions that implement the components, Component Controller and Component Logic. Even with the extensions we cannot control the EDI execution to fully emulate the behaviour of a normal Wi-Fi Direct connection. To solve this problem, Termite proposes a new design for the Component Controller and Component Logic. The modelling of the Emulated Disk Image must incorporate both controllers goals and the interaction with the Master Process. Although the importance of the emulated disk image modelling is easy to understand, it is also important to keep in mind its architecture and behaviour. It is imperative to keep intact the interactions between the emulated image and the applications. A minor alteration in this interaction would be enough to change the expected behaviour of the image, creating complications to the application and subsequently to the user of the Termite system.

With the purpose of simplifying the solution architecture, we propose a different approach to the Controller Logic and Controller Component description merging them into a single Controller, the Termite Controller. Interactions between the Controller and the Master Process remain the same, as well as the

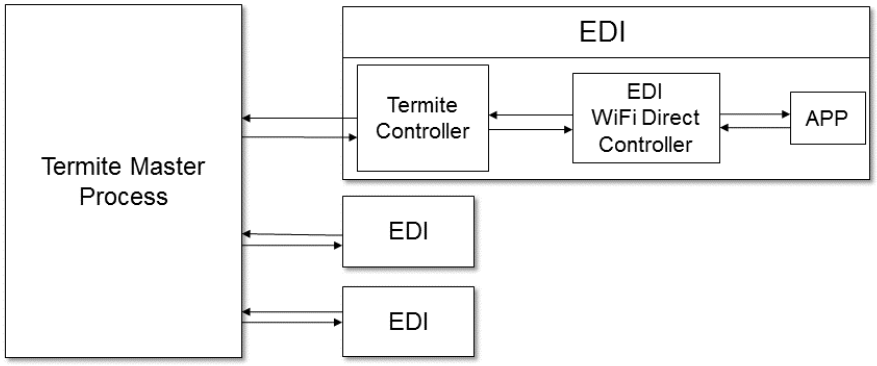


Figure 3.3: Interaction between the Master Process and the Controller inside the Emulated Disk Image

ones between the Emulated Disk Image and Termite. The main changes are explained on the Emulated Disk Image architecture description and how the Termite components are added to it, as shown in Fig.3.3.

To better understand this new concept of Master Process/Controller-Application the next subsections further detail each component and how they contribute to the solution architecture.

3.2.1 Client-Controller Interaction

Before understanding the chain of events leading to the system execution it is necessary to understand every solution component. First we have the Master Process, as explained on the previous section, the Master Process works as coordinator of the emulated images. Whenever there is an event set by the controllers, it is the Master Process responsibility to ensure the next step is correct. Thus ensuring the emulation state. Next we have the controllers, responsible for managing the EDI state, collecting and distributing events generated by the Emulated Disk Image Wi-Fi Direct Controller(EDI-WDC) or Master Process. After this a new concept is introduced: the Client. The Client has the same functionality and responsibility than the Emulated Disk Image Wi-Fi Direct Controller. The emulated image Wi-Fi Direct Controller was responsible for the coordination with other devices in a real Wi-Fi Direct environment . Now the Client is responsible for replacing the EDI-WDC, maintaining all the same interactions but working together with Termite. Finally, there is the Application. The application has no context about the difference between the Client and the Emulated Disk Image Wi-Fi Direct Controller.

The interaction between the Client and the Controller happens on the context of the event chain. To better understand how this interaction works, we must remind how the Emulated Disk Image Wi-Fi Direct Controller works. Whenever there is a change in the network, the EDI triggers an event. This event is managed by the Emulated Disk Image Wi-Fi Direct Controller. After receiving the event the EDI-WDC takes the best action, that is, to retain the event or to forward the event to the application. After the event is forwarded to the application, it is the application responsibility to decide what is the best action. As

explained before, these events are produced by alterations in the network, in this case the Wi-Fi Direct network. If there is no Wi-Fi Direct network, it is expectable that no events reach the EDI-WDC, and consequently, the application. Therefore, in this scenario, when the emulated network changes, it is Termite's responsibility to produce such events. In Termite 1.0 the EDI extensions received those events and forwarded them to the application, but that is not viable any more, according to our requirements. The connection from Termite to the application would risk the EDI-WDC - Application interaction. Then we reach the core functionality of the Termite Controller.

According with this new interaction design, we introduced the Termite Controller. The Termite Controller receives the events sent from the Master Process and forward them to the Client component. This interaction compels the Controller to be able to translate Termite Events into EDI Events, which is the main objective of the Controller.

3.2.2 Controller Event Management

As previously explained, the Controller receives those events and must forward them to the Client. These events contain important information about the network and the emulated image state. Therefore the event management requires a translation which is not trivial, because they are not only new events, but they carry information about the emulated network, or the emulated image. The Controller has the responsibility of translating the received events.

One of our goals is to have an EDI specific level abstraction. This means that there are numerous ways for the Client to operate. To support this abstraction the Controller must be able to translate Termite Events. The translation is dependent on the emulated image, which forces the Client to be EDI specific, too. The event chain does not depend singularly on the Client. This implies the Controller must be also EDI specific. We call this dependency: Specific EDI Interaction.

After understanding the concept of Specific EDI Interaction, we can take some conclusions about the integration of the EDI into the Termite system. Since the translation responsibility begins with the Controller and propagates to the Client, there is a certain independence of the EDI, regarding the Master Process. This means that the Specific EDI Interaction, provided by the new Controller Event Management design, is transparent to the Master Process.

After analysing the interaction between the Controller and the rest of the system, we can observe the independence between the Controller/Specific EDI Interaction and the Master Process. This takes us to the next topic: how the Master Process relates with the rest of the system.

3.2.3 Termite Master Process - EDI Interaction

The Master Process is a core component of the system with several key responsibilities for the overall system execution. One of them is the interaction with the Termite Front-end. The Master Process receives the developer settings and transforms them into the correct events to be propagated. The other responsibility is to make sure the Emulated Disk Images Controllers receive those events. But the Master Process does not interact only this way. There is also an interaction between the EDI Controllers and the Master Process that leads to event triggering.

The abstraction between the Master Process and the EDIs is critical for the system, to fulfil some of the requirements described in Section 1. Adding this abstraction layer improves the system portability and scalability. This feature allows the developer to use multiple platforms to execute multiple EDIs. To do that it is necessary to have a connection between the EDIs and the Master Process, and the EDIs presents a similar behaviour while the application is running. The second requirement is more complex than the first. When the same application interacts with different Clients but uses the same input it must be expectable that, under the same conditions, the output would be equal. Regarding the platform that runs the EDI, there must be a connector that ensures correct interaction with the Termite system. This connector was addressed on previous section.

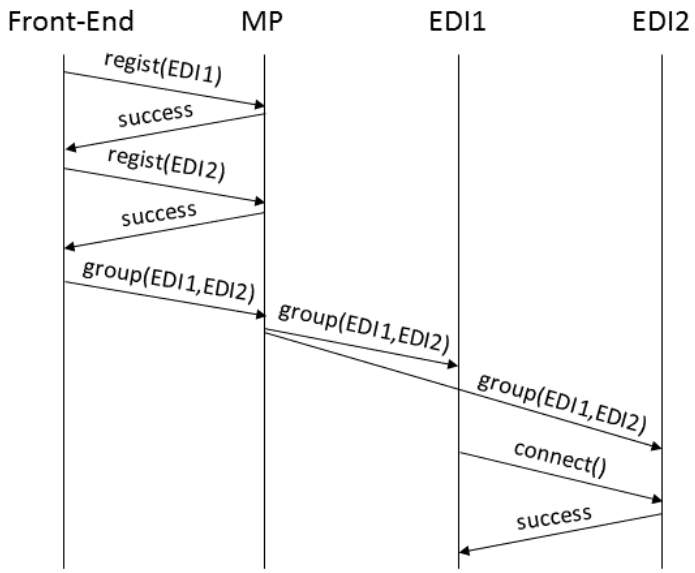


Figure 3.4: Creation of 2 nodes network in Termite

MP plays an important role creating the connection between EDIs. The events triggered by the MP contain the necessary information in order EDIs can create channels between them. After receiving the Connection event, the developer's chosen EDI will connect to the second EDI setting up a communication channel as we can observe in Fig.3.4 example.

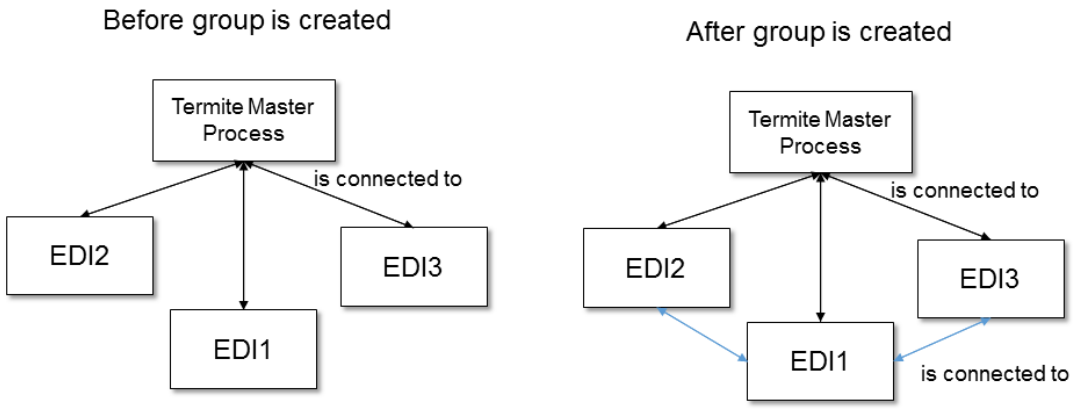


Figure 3.5: Connection between EDIs

After both EDIs being connected, the Master Process is not responsible for the interaction between them any more, as we can understand from Fig.3.5. The MP only interacts again with either one of them in two cases. First, an EDI loses MP connection. In this case the Master Process triggers a new event to the EDI that is still connected. The second one is when the developer issues a command over the emulated network. In this case the Master Process receives the command and sends the correct event to one or both of them.

As explained before, the network will not be limited to two nodes since one of the requirements is scalability. So, how will be the protocol applied and the network arranged? Let's observe how will the protocol evolve when there are 4 nodes to be connected, as described in Fig.3.6. In this example, we connect two more nodes after having already two nodes configured, as seen in Fig.3.4. The Front End registers all the EDIs to the MP, and after that, the MP informs the main network node. This node is called Group Owner(GO) and is responsible for the connections between the other nodes as seen in Fig.3.5.

This network arrangement does not allow peer-to-peer connections that do not respect the connection Node-Group Owner. This brings us to our next topic, Message Forwarding.

3.2.4 Message Forwarding

After the EDIs being connected, it is time to exchange information among them. As seen on the previous subsection the EDIs are not connected to each other. Instead they are connected to the GO. Using Fig.3.5 example, if the EDI2 wants to send a message to EDI3, then it must send the message to the GO node, and the GO node must forward the message to EDI3. This is not static protocol in message forwarding. The application has the freedom to chose alternative forwarding mechanisms which are not

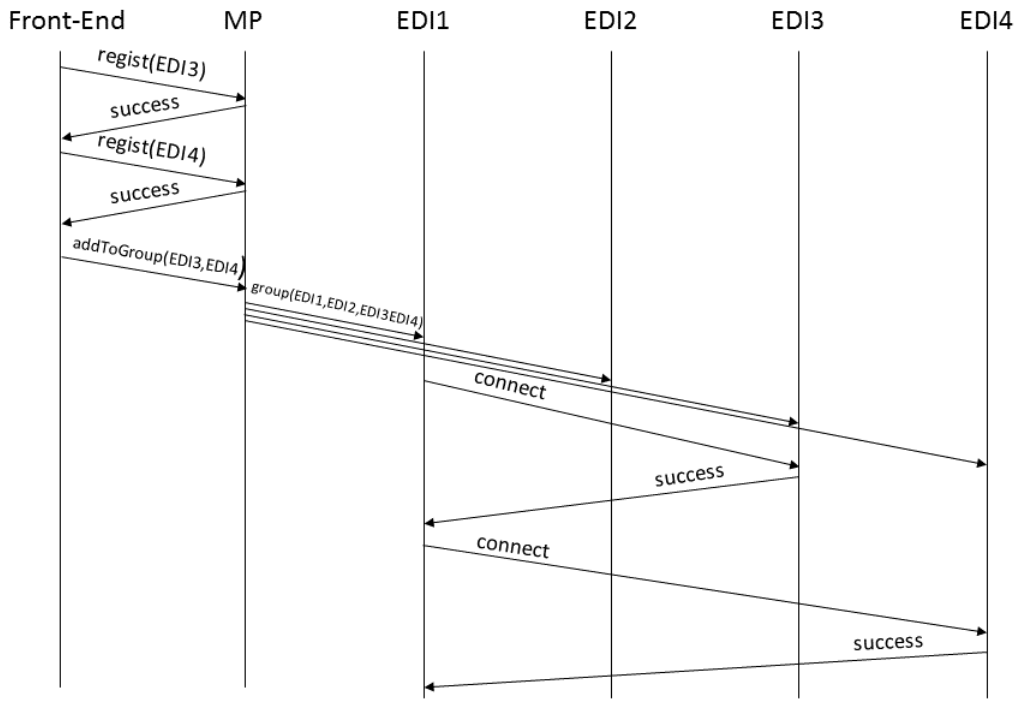


Figure 3.6: Connection between EDIs

bounded to the system, except for the Node-Group Owner connection rule which prevents node-to-node connection. Another solution could be the use of network broadcasts. In this case the node decides if he captures or ignores the message.

A problem created by this mechanism is how the emulated image should deal with sending and receiving of messages. As the developer has full control over the application that uses Termite, it makes sense to let him decide the best channel. After receiving the connection information from the Client, it is the application responsibility to ensure the communication between EDIs. To support this portability the Client must be enforced to support the communication options which are supported for that specific Emulated Disk Image. It is also important to refer that all the EDIs must support connection to each other in real scenarios to ensure the emulation process, while respecting the GO-Node rule.

Termite has the responsibility of ensuring that the messages are correctly send and received among the devices. After entering a stable state, the emulation proceeds with message forwarding inside the network until there is a new event generated by any one of the system components. Some examples are: a new connection or any EDI connection loss.

3.3 Termite Use Case

For a better understanding of Termite architecture, this section describes a simple use case of the system. We start describing the background and the desirable goals. The use case is explained from a Termite user point-of-view and will aggregate all the components described before.

The goal of the use case is to run an application that, through the emulated Wi-Fi Direct, sends a broadcast to all network nodes, identifying the device. The application also receives other devices broadcasts and displays the name of every device in the network to the user. To do this, Alice has already the Termite system ready in her computer. She also started 3 emulators with duplicated EDIs containing the Termite necessary modelling.

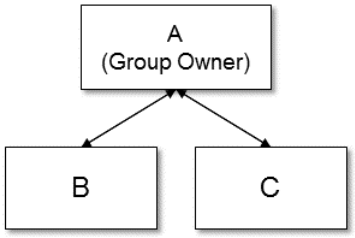


Figure 3.7: Group Owner-Nodes Connection

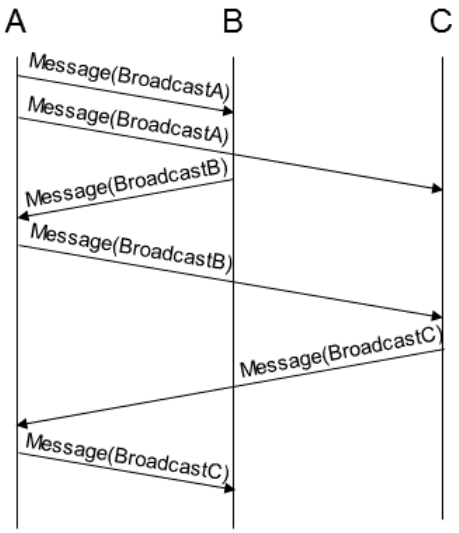


Figure 3.8: Broadcast Protocol example

To begin, developer Alice starts Termite Devel Client where she registers 3 EDIs. The Devel Client starts

the Master Process and registers the EDIs. The EDIs are registered with the names ”A”, ”B” and ”C”, which were chosen by Alice. After this step, Termite is ready and the developer can start managing the network. Before creating groups, Alice deploys the application that starts the Termite Module inside the Client, preparing the Controller for event capturing and managing, even without her intervention. When Alice begins to use the Devel Client to manage the network, all EDIs are ready and waiting for MP to send any necessary event. The developer issues a command through the front-end that groups all the three devices under the ownership of device ”A”, this means that ”A” is the GO. The Master Process sends the information about this group creation to the Termite module inside each EDI. After receiving this information, the Controller triggers an event which is forwarded to the application notifying that the device is inside a new group. When all the devices receive the information, the network graph should be similar to the one presented in Fig.3.7. Following the network creation, Alice uses the application functionality to send a broadcast identifying the device that sent the message. The protocol used by the application to broadcast the messages is described in Fig.3.8. To finish the usage of the application, Alice asks each device to print the list of other nodes presented in the network, which printed the output shown in Fig.3.9.

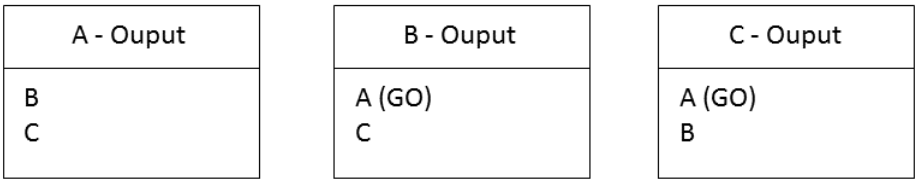


Figure 3.9: Usecase Ouput

3.4 Summary

In this chapter we presented an overview of Termite Architecture, an emulation test-bed for encounter networks. To set the foundations of Termite, we presented the background that led to the development of this project. Termite 1.0[53] was explained and its architecture was described in the first section. After this, we described the architecture of the solution that allows us to fulfil the requirements stated in the first chapter of this document. We began to explain where the identified flaws of Termite 1.0 were. The main focus of the new architecture is the Emulated Disk Image modulation that replaced Termite 1.0 Emulated Disk Image extensions. We described and explained each component added to the architecture, as well as the interactions among them. To ensure the comprehension of Termite system chain of events, we added a use case scenario that focused primarily on the point-of-view of the developer that uses Termite to test and debug his Encounter Based Network Application.

Chapter 4

Implementation

In this chapter, we present the implementation details of Termite, an emulation test-bed for encounter based networks applications according to the architecture described before. On Section 1 we start providing a high-level overview of the technologies used and the components that are part of Termite. Afterwards, Section 2 sets the background by describing some of the most important flaws of Termite 1.0 implementation. In Section 3, we present the Termite implementation by focusing the most important challenges faced on developing the prototype. Finally, in Section 4, we summarize the implementation.

4.1 Termite Overview

In this section we explain the usage of Wi-Fi Direct and establish a connection with Android. After that we explain some of the components that are part of Termite.

4.1.1 Android and Wi-Fi Direct

Termite implementation uses Android and Wi-Fi Direct technology. Android is the largest used operating system in the world for mobile devices[56], representing more than 6 mobile devices running Android per 10. Also, being an open source system, through the Android Open Source Project[57], clearly softens the choice to implement a prototype of this project.

Wi-Fi Direct[58] is a technology that improves the Wi-Fi capabilities providing an easy way to connect multiple devices that are Wi-Fi Direct enabled. One of these devices must have a Wi-Fi CERTIFIED Wi-Fi Direct to serve as hotspot. This happens because normal Wi-Fi devices do not support such functionality. Currently, there are 8660 products which are Wi-Fi Direct certified, among those 2917 are mobile devices[59].

Google has developed an Android library that fully implements the Wi-Fi Direct functionality. This library, included on Android Framework, was released in version 4.0.0, Android Ice-cream Sandwich.

Every mobile device that is running this version or higher, and was not modified by the manufacturer, supports Wi-Fi Direct connections. However, as explained in the previous paragraph, some devices may not support the role of Access Point.

This project will focus in the Application Framework Layer on Android Operating System. The core implementation of the Wi-Fi P2P framework, available to all developers, needs to be and reprogrammed to support emulators and off grid communication through local or wide networks, for example an emulator running in a local machine must be able to communicate with an image running in a cloud server.

4.1.2 Android Framework

Android provides an API for application development that takes advantage of Wi-Fi Direct. But this API is limited to the use of physical devices, becoming hard to emulate interactions between virtual devices. To overcome this obstacle, Termite 1.0 proposed the use of a specific API to support Wi-Fi Direct interaction between virtual devices. As explained before this solution has several flaws regarding this project requirements, therefore we propose another solution.

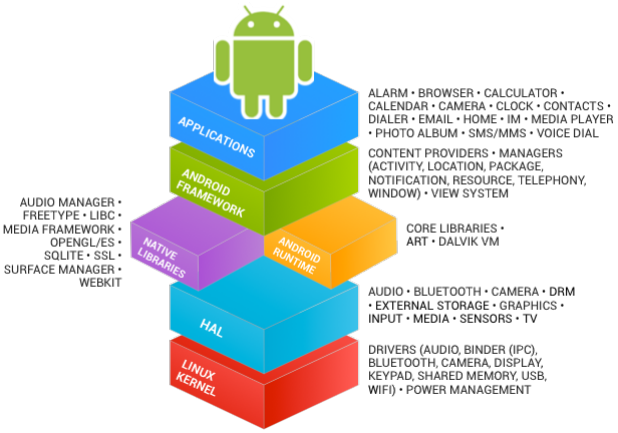


Figure 4.1: Android Layers.

Android API management is made in the Android Framework layer. This layer is between the Application layer, and Android runtime and Native library layers. The Android Framework is most often used by application developers. This framework provides many high-level services to applications in the form of Java Classes. Application developers are allowed to use these services in their applications.

Android Framework implements Wi-Fi Direct API. Fig. 4.2 presents a code snipe of the WifiP2pManager Class implementation [60]. This class is responsible for connections management that use the Wi-Fi Direct channel.

For Termite implementation, the core idea is to change the implementation of the Wi-Fi Direct frame-

```
911     /**
912      * Initiate peer discovery. A discovery process involves scanning for available Wi-Fi peers
913      * for the purpose of establishing a connection.
914      *
915      * <p> The function call immediately returns after sending a discovery request
916      * to the framework. The application is notified of a success or failure to initiate
917      * discovery through listener callbacks {@link ActionListener#onSuccess} or
918      * {@link ActionListener#onFailure}.
919      *
920      * <p> The discovery remains active until a connection is initiated or
921      * a p2p group is formed. Register for {@link #WIFI_P2P_PEERS_CHANGED_ACTION} intent to
922      * determine when the framework notifies of a change as peers are discovered.
923      *
924      * <p> Upon receiving a {@link #WIFI_P2P_PEERS_CHANGED_ACTION} intent, an application
925      * can request for the list of peers using {@link #requestPeers}.
926      *
927      * @param c is the channel created at {@link #initialize}
928      * @param listener for callbacks on success or failure. Can be null.
929      */
930     public void discoverPeers(Channel c, ActionListener listener) {
931         checkChannel(c);
932         c.mAsyncChannel.sendMessage(DISCOVER_PEERS, 0, c.putListener(listener));
933     }
934
```

Figure 4.2: Android WifiP2pManager Class Implementation example [60].

work developed by Google, and replace it for Termite implementation. This implementation will allow both physical and virtual devices to support Termite, and later on to establish connection between the two devices. Also, not having a specific custom API, the developer can use the specified Android Wi-Fi Direct API without knowing that the implementation was modified. This will give Termite a transparent look from the application perspective. When the application is exported to a free Termite environment, with the stock Android Framework implementation, the application should run smoothly without any issues to the developer.

4.1.3 Platform Connectors

The platform connectors allow Termite to deploy a wide variety of virtualization platforms. The connectors are software components that support the platform API and work as plug-ins for the Admin and Devel Clients. Termite presents an implementation in an isolated Python module which is shipped with the Command Line Interface(CLI). If it is necessary to add new platforms to Termite, it can be achieved by adding dynamically new connectors to the previous version of the CLI without the need to rebuild the full framework.

The connectors job is to accept commands from the Admin and Devel Clients APIs translating them into virtualization back-end commands using the platform specific API. Because of the changes in the Termite API and the re-allocation of the Android Framework implementation, these connectors were updated to support the new API, while providing the necessary compatibility to execute the clients commands.

4.2 Background

In this section we describe Termite 1.0 prototype implementation. To better understand the improvements and the issues faced in Termite implementation, it is crucial to understand Termite 1.0. This section is divided in three subsections. The Design subsection describes the implementation architecture and will relate it with the conceptual architecture described in the previous chapter. The second subsection will approach the interaction between Termite 1.0 front-ends and the Emulated Disk Image. Termite Library subsection is focused on Termite implementation inside the emulated device, which is the main focus of Termite 1.0 improvements.

4.2.1 Design

In this subsection we will go deeper into Termite 1.0 architecture explaining Termite 1.0 interactions inside the emulated image. One of the concepts we present is the capability of broadcasting inside the Emulated Disk Image. Instead of communicating directly to a component, Termite 1.0 uses broadcasting to send messages inside the mobile device system. This is not a specific component of the technology used in the prototype implementation, but later on in this chapter we will relate it with the equivalent Android.

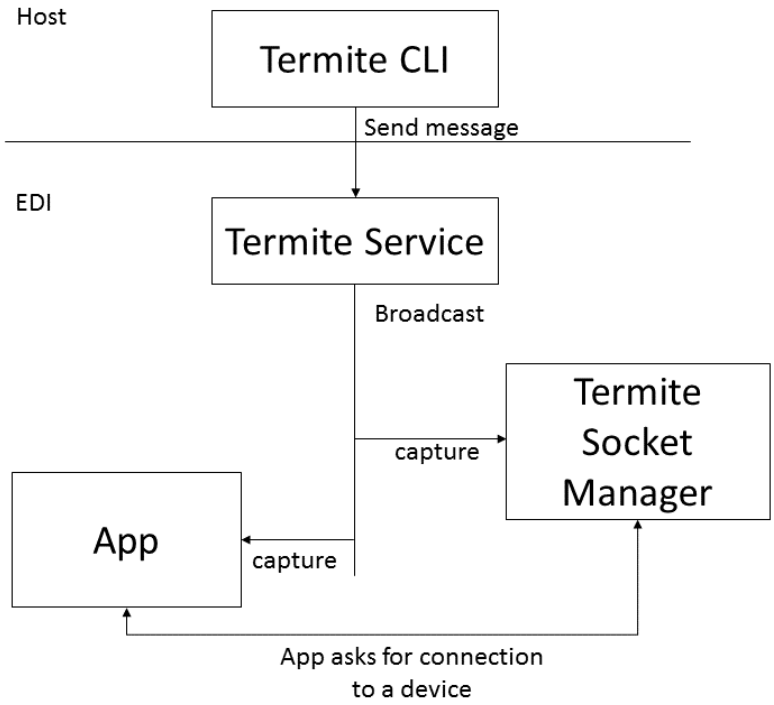


Figure 4.3: Termite 1.0 Implementation Design

In Fig.4.3 we present a scheme of the Termite 1.0 implementation design. In this figure we have two main environments, the ”host” that is where the Termite front-end is running, and the EDI, that is the emulated image running in a emulator that can be local or remote. The Termite CLI is the Command Line

Interface, where the developer controls the emulation process. The Termite Service is running inside the emulated image and it is started by the application. Both Termite Service and Termite Socket Manager are components of the Termite Library. Termite Library is the extension provided to the emulated image.

Lets take for example the case where the developer wants to connect two mobile devices in the same network. When the command is issued in Termite Command Line Interface, it sends a message to the Termite Service which is connected to Termite front-end. The Termite Service must receive that message and translate it in a termite event to be forwarded to the Application. After creating the event it must be broadcast for different components.

4.2.2 Termite Library

Termite Library is where all the extensions for the emulated image are implemented. The specified component controllers in section 1 of chapter 3 are implemented in this library. The Termite Library is loaded into the emulated image with the application binary. The application is dependent of the library to be able to execute.

In order the application uses Termite 1.0, the developer has to code it based on Termite API[61]. A Wi-Fi Direct application coded for Android would use the Android Framework Wi-Fi Direct API. Thus, an application made for Wi-Fi Direct had to be entirely recoded to use Termite 1.0. Although the Termite 1.0 implementation is based on Android Wi-Fi Direct implementation, the Termite Library has no connection to the Android Framework, and simply emulates it.

The lack of support to test applications built with Android WiFi-Direct API served as a motivation to the implementation of a new prototype that is described in the next section.

4.3 Termite Prototype

The purpose of this section is to describe the Termite prototype. The Termite prototype was developed on top of Android Marshmallow(6.0.1). We chose this version because it was the most recent supported version in Android Open-Source Project(AOSP)[57]. Although we used a fixed Android version, the process of porting Termite to a future Android version would be trivial, as the structures involved are present in the different versions. The Android tag present in AOSP is ”android-6.0.1_r17”. The Android custom image was compiled in a virtual machine running Linux Ubuntu 14.04 and tested in an emulated Nexus 6. All devices were emulated in the Android Virtual Device(AVD)[62].

In this section we present the prototype design and explain the major challenges found in the imple-

mentation.

4.3.1 Design

Termite prototype design was developed having in mind interactions with the Android Wi-Fi Direct module. It is important to maintain the Wi-Fi Direct API as it was, so that applications can be tested in Termite without the need to change their code.

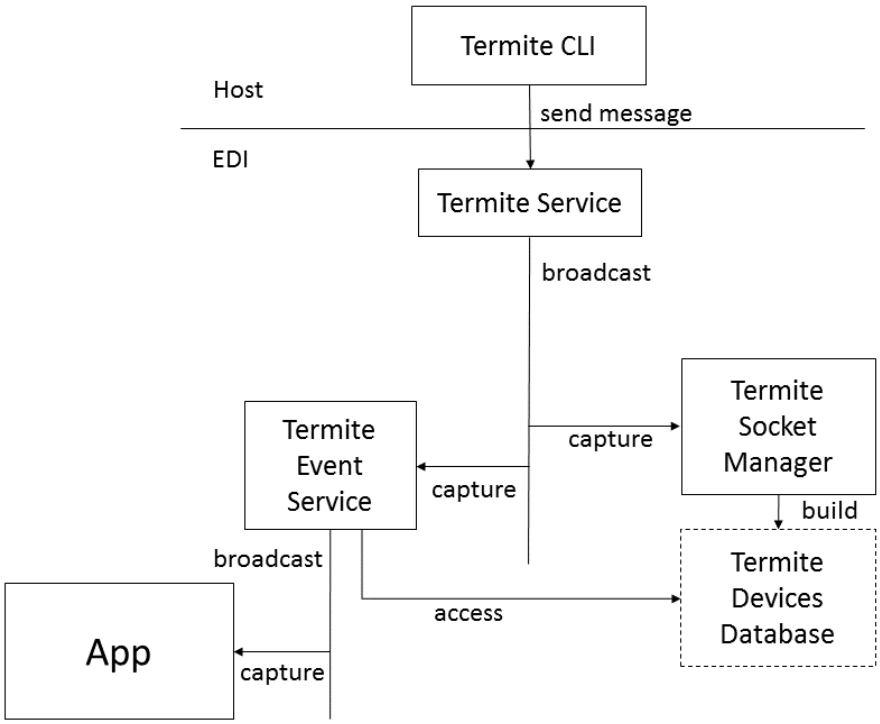


Figure 4.4: Termite Implementation Design

In Fig.4.4 the Termite Front-end still connects directly to the Termite Service running in the emulated device. After the Termite Service receives the message, it is its job to broadcast a Termite Event with the information correspondent to the message. Two components have the responsibility to capture this events. Termite Socket Manager captures the event and uses the information to build a database with the devices connected in the network. Termite Event Service captures the event and translates the Termite Event to a WiFiP2p Event, which is a native Android event. This translation is made, so that, the application can receive the broadcasts exactly as it would be transmitted by the Android Wi-Fi Direct module. To be able to do this translation, the Termite Event Service accesses the database that the Termite Socket Manager builds. When the WifiP2p event is broadcasted, the application captures it and proceeds with its normal execution.

With this new Termite design, the application can use Termite, as it would do in a native Android Framework implementation. Unlike Termite 1.0 implementation, which required a specific API, Termite imple-

ments the required system transparency to test native applications. This transparency improves Termite usability.

4.3.2 API Integration and Update

One of the explained problems in this chapter is the usage of Android Wi-Fi Direct API. The API must be static so that applications developed using Android Wi-Fi Direct can be tested in Termite without the need to change the application code. The Android image must be modelled to support Termite and to do this the Android Framework Wi-Fi Direct module must be recoded. While adding Termite code to the Android Framework the Android API was constantly updated with Termite methods that could be visible by developers. One of our requirements was to create a transparent system, and to do this Termite must respect the diagram in Fig. 4.5.

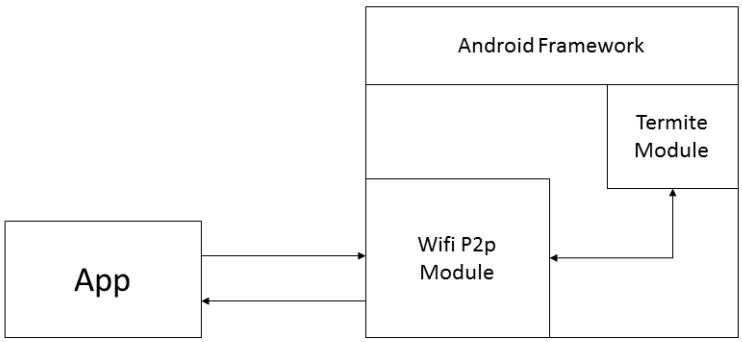


Figure 4.5: Termite API Integration

To solve the API problem we had two solutions. The first was to hide all the new methods added by Termite to the API, creating the appearance that the Android Framework remains unchanged. The second option was to keep Termite back-end methods visible to the programmer. The first option had no utility to the developer. So we decided to maintain Termite methods visible. The main advantage of such a decision was that by keeping Termite functionalities visible by developers, we were giving them more tools to debug their own applications. Although the application was built with Android Wi-Fi Direct methods, the programmer could use Termite back-end to help him debug the application. An example could be use the Termite Socket Manager database to be aware of the available devices within Termite emulated network.

4.3.3 Flow Control - Android Broadcast

As seen in Fig.4.4, Termite and the Wi-Fi Direct module use broadcasts to share information to other components. In Android there is a simple method to use these broadcasts. Each application can register a BroadcastReceiver[63] with the type of broadcasts they want to capture. In this case, Termite Event Service and Termite Socket Manager need to register a broadcast receiver to capture the termite events,

while the application needs to register a broadcast receiver for Wi-Fi Direct events.

Broadcasts serve both as execution control and as data forwarding among different components. These broadcasts were made for asynchronous messaging among the Android system and the applications. Termite uses the broadcasts as the main mechanism for communication among modules. As Termite is transparent to the application, the events broadcast need to be equal to native Android Wi-Fi Direct events. Termite has the responsibility to translate termite events, as well as the information carried within. The broadcast receivers are registered in the application context. Therefore, it is necessary to have access to an application to receive events.

Registering Broadcast Receivers inside Termite is a problem, once Termite is not an application. As Termite is implemented in the Android Framework layer it is not possible for it to register broadcast receivers. To solve this issue, it is necessary to have access to the application context, and to do this we took advantage of a component called Emulation Controller.

4.3.4 Emulation Controller

When booting physical devices with Termite implementation it became important to be able to control which module was running the Wi-Fi Direct components. Although it appears a simple task, to turn on and off the Termite emulation, it became a challenge on how to implement this functionality. To solve this problem we found three solutions, one depended on the application and the other two on the Android system.

The first solution was to add a parameter on the Wi-Fi Direct module API, to ask the application, if it would want to use the native or Termite implementation. The main problem, in this solution, was that the API would have to be changed and Termite would no longer be transparent to the application. The second solution, which has two alternatives, was to add a controller inside the Android Framework that would serve as a switch to the emulation. With this design there was no need to change the API, but there was no way to access the controller directly. Either one of two alternatives could be used, create a Termite application with the single purpose of controlling the emulation or to add a switch to a standard Android menu using a built-in application to access the application context.

The problem was solved by adding the Emulation Controller component to the Android Framework, and to create a switch inside the Android Settings Menu to turn Termite emulation on or off. Fig.4.6 shows the switch implemented inside the menu Settings>>Wireless & Networks>>More.

As mentioned in the previous subsection the broadcast receivers need application context to be registered. The emulation controller switch is inside the Settings application. So, we developed a solution

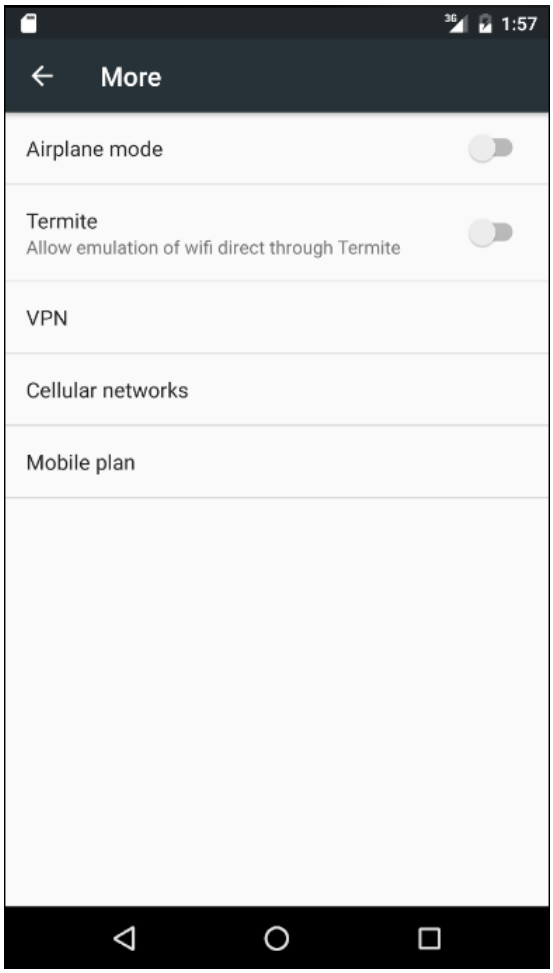


Figure 4.6: Screenshot of Settings Menu

that when the Termite emulation is enabled uses the Settings application context to register all the Termite services. This allows us not only to register all the needed broadcast receivers but also to unregister them when the emulation is turned off. This solution is implemented by the Termite Enabler, which is a component inside the Emulation Controller.

4.3.5 Socket Redirection

One of the most important functionalities of Termite is connecting two emulated devices. To be able to connect two devices Termite needs to complete several steps that were already described in this document. One of these steps is converting of virtual addresses to real addresses. Virtual addresses are the ones used by applications to specify the devices, real addresses are used by Termite to identify virtual devices, and CV addresses are used locally to represent the emulated image.

Fig.4.7 represents the process of converting addresses in Termite 1.0. The most important change in the process of establishing connections is the removal of Termite Socket Manager. This component was

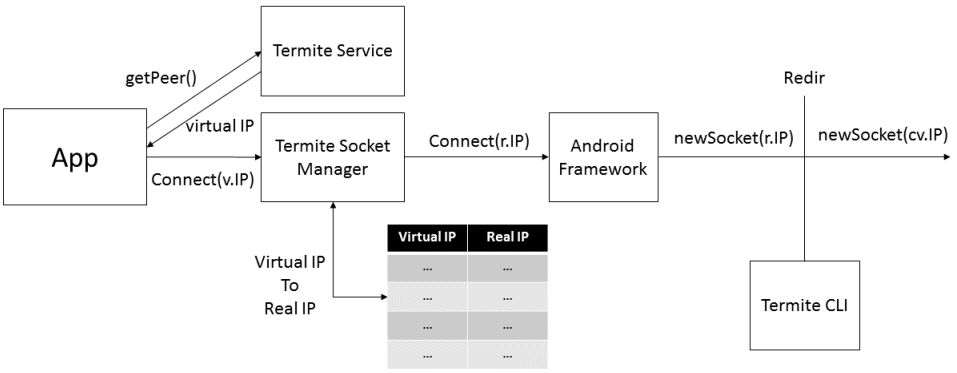


Figure 4.7: Socket Redirection in Termite 1.0

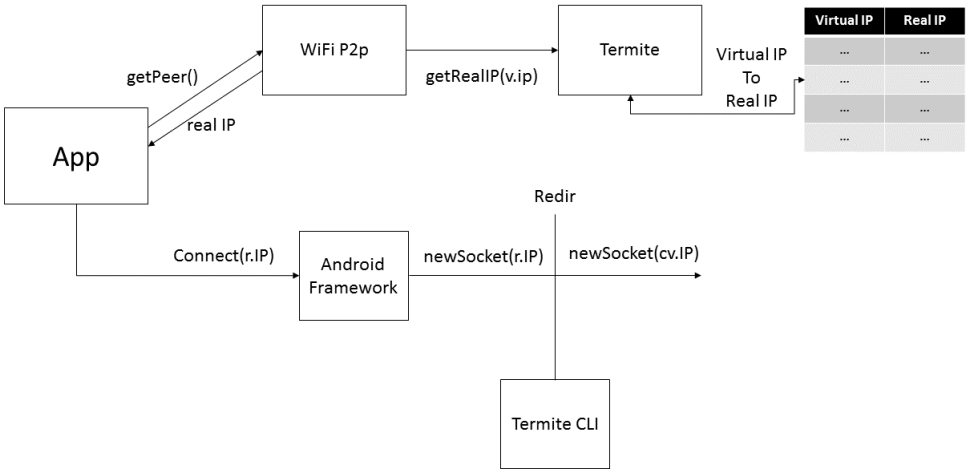


Figure 4.8: Socket Redirection in Termite

responsible for creating and maintaining connections between devices. In Fig.4.8 we can observe that Termite is now condensed to one single component. The unique goal of Termite interaction in this phase is to convert the virtual address into a real one. The stock Android Wi-Fi Direct implementation expects the developer to handle the creation and management of connections. Therefore we need to provide the real address to the developer, so he can implement all the connections workflow in his application.

In the figures we can also observe the conversion from real address to cv address. We can also notice that this translation is made by the Termite Front-End. When the Termite Front-End registers an emulated device, the Front-End uses the connector to issue a redirection command. In the local case, the local connector connects to the emulator port through a Telnet connection and issues the commands necessary to redirect every connection made with the real addresses to the cv addresses. These commands add the redirection addresses to the redirection table present in the emulator.

4.3.6 Connector Update

Android images, development kits, tools and IDE are not static and are constantly updated. These updates can sometimes have impact in protocol, accesses, etc. A problem found during the implementation of Termite was an update in the access protocol of emulators, rendering the protocol obsolete. The Termite Platform Connector used a Telnet connection to access the emulated platform, in this case Android Virtual Device. To increase security in the Telnet connections made to the emulators, Google added Token Verification in the connection protocol. This token is created in the user local folder when the emulator is initiated. To support the change made in the protocol an update must be done in the local connector.

When the Telnet connection is established, the emulator replies with the token location string. We reimplemented the connector to use this string to access the token and provide the correct authentication. After the newly add authentication, the protocol remains the same, having the connector issuing commands to add the redirection rules to the emulator.

4.4 Summary

In this chapter, we presented our implementation of Termite system. Termite was built for the Android platform(Marshmallow version). To be able to implement the necessary requirements, the Android source code had to be modified. These modifications required adding the entire Termite functionality to Android Framework, as well as changing Android Framework Wi-Fi Direct module source code.

We presented the most challenging obstacles found when implementing Termite in the Android Framework layer. To solve these challenges, we describe each choice done and justify the options made in the design and implementation of Termite.

In the next chapter, we present an evaluation of our system, as well as comparisons to real world scenarios.

Chapter 5

Evaluation

In this chapter, we present the evaluation and corresponding results collected throughout this work. We start showing the impact of network creation and deployment. Afterwards, we evaluate the functionality and usability of Termite implementation. We also deliver a performance evaluation on the new Termite Emulation Disk Image and compare it with real environment results. Finally, we summarize every component of the evaluation and present some conclusions.

For the presented tests in this chapter, we used an Asus laptop, featuring a quad-core up to 3.5 GHz CPU, 8 GB of RAM and 750 GB of memory. In this laptop, we emulated two Nexus 6 smartphones, featuring 2 GB of RAM, 200 MB of internal storage and 100 MB of SD Card. Both emulators were flashed with a build of Android 6.0.1 AOSP patched with Termite code. We also used two Nexus 6 smartphones, featuring a quad-core 2.7 GHz CPU, 3 GB of RAM, 64 GB of memory and 802.11 Wi-Fi interface.

5.1 Network Creation and Deployment

One of the major components of Termite is the emulated network management. As explained, in this document, the network emulation is made by Termite, but the user needs to specify the network topology. This topology can include several nodes spread across more than one machine, not necessarily in the same physical location. Thus, Termite network deployment is an important component and has several implications in the system correct execution, performance and usability.

As described in the Architecture and Implementation chapters, the work made in this thesis focus the Termite Emulated Disk Image modelling. When compared with Termite 1.0, the deployment and network managing is roughly the same. However, the connection process between the Termite Front-end and back-ends was improved in the form of a platform connector update described in chapter 4, section 2. Therefore we think the Termite 1.0 deployment evaluation[53] should be accurate when compared to our prototype.

Table 5.1: Workload execution times in seconds

		Termite 1.0	Termite
Workload 50	min	158.93	156,29
	avg	159.99	157.61
	max	162,13	157,52
Workload 100	min	312.48	309.10
	avg	314.25	310.01
	max	315.66	310.85
Workload 250	min	772.81	770.13
	avg	773.27	771.56
	max	774.08	771.81

However, to prove the idea that Termite network management performance compares with Termite 1.0 evaluation, we developed a deployment test. The goal of this test is to prove that Termite network performance is at least identical to Termite 1.0. To set to such a conclusion, the test is composed by successive creation and deletion of a network between two nodes in a local machine. The Termite Script used to perform the test is available in the Appendix A of this document.

To produce the results we divided the tests in three different workloads. First, we repeat the process 50 times, followed by a workload of 100 creations and deletions. To finish, we use the same loop but 250 times. We repeated the test 10 times for each workload and registered the execution times of each one.

As seen in Table 5.1, Termite has a performance not worse than Termite 1.0. In all the workloads Termite performance matches Termite 1.0, always having a small advantage over it. We can observe that in all the workloads Termite has a 1 to 3 % margin over the Termite 1.0 values. The entire result set is available in the Appendix B of this document.

We can conclude that in terms of network deployment and management, Termite performs well and follows closely the evaluation presented in Termite 1.0. Therefore, the changes in the platform connector, and the improvements made in the Emulated Disk Image, did not have relevant impact on Termite network deployment performance.

5.2 Functionality and Usability

Allowing developers to test their Encounter Based Network Applications without changes in the application code is the major contribution of this thesis. As seen in this document, the options previously available to developers present substantial issues. The lack of a dedicated test-bed to assist developers to test and debug Encounter based Network Applications, led to the development of Termite.

When compared to Termite 1.0, Termite allows the developer to use the native code that was developed for Android Wi-Fi Direct API as it is implemented. This is a great improvement and brings a lot of advantages to the developer. The need to rewrite the code to use a different API is no longer an obstacle

to the application test, as it was in Termite 1.0. In this section we address the changes between native and Termite 1.0 application code, to prove Termite improved functionality and usability when compared with Termite 1.0. We also describe the advantages of using native code during the test and debug process, as well as the application deployment cycle steps that are significantly improved.

Android WiFiP2p Library

```
WifiP2pManager mManager;
Channel mChannel;
BroadcastReceiver mReceiver;
...
@Override
protected void onCreate(Bundle savedInstanceState){
    ...
    mManager = (WifiP2pManager) getSystemService(Context.WIFI_P2P_SERVICE);
    mChannel = mManager.initialize(this, getMainLooper(), null);
    mReceiver = new WiFiDirectBroadcastReceiver(mManager, mChannel, this);
    ...
}
```

Termite Library

```
private SimWifiP2pManager mManager = null;
private Channel mChannel = null;
private Messenger mService = null;
...
private ServiceConnection mConnection = new ServiceConnection() {
// callbacks for service binding, passed to bindService()

      @Override
      public void onServiceConnected(ComponentName className, IBinder service) {
            mService = new Messenger(service);
            mManager = new SimWifiP2pManager(mService);
            mChannel    =    mManager.initialize(getApplication(),    getMainLooper(),
null);
            ...
      }
```

Figure 5.1: Code snipe of Android WifiP2p and Termite 1.0 Initialization.

In Fig.5.1 we can observe the differences between initializing a native application and a Termite 1.0 application. The big difference in this case is the initialization of the Wi-Fi Direct manager. While in android the Wi-Fi manager is created within Android Framework and initialized in the application, in Termite 1.0 the manager needs to be created and initiated within the application context. Another difference is the events received by the application. In the Fig.5.2 we can observe that there is not a direct correlation between the events registered by native android and by Termite. These events control the application execution flow as previously described in chapter 4. Since the events behave differently, the application execution needs to be changed to produce the same result as it would have in native code.

As described in the previous paragraph, the event discrepancy between both implementations bring problems to the developer to swap between both frameworks. In Fig.5.3 and Fig.5.4 , we can see the difference between both implementations, and observe that there are differences in the objective of each event. The translation of these events is not trivial. Section 2.7 of Chapter 4 described the main prob-

Android WiFiP2p Library

```
IntentFilter mIntentFilter;
...
@Override
protected void onCreate(Bundle savedInstanceState){
    ...
    mIntentFilter = new IntentFilter();
    mIntentFilter.addAction(WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION);
    mIntentFilter.addAction(WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION);
    mIntentFilter.addAction(WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION);
    mIntentFilter.addAction(WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHANGED_ACTION);
    ...
}
```

Termite Library

```
IntentFilter filter = new IntentFilter();
filter.addAction(SimWifiP2pBroadcast.WIFI_P2P_STATE_CHANGED_ACTION);
filter.addAction(SimWifiP2pBroadcast.WIFI_P2P_PEERS_CHANGED_ACTION);
filter.addAction(SimWifiP2pBroadcast.WIFI_P2P_NETWORK_MEMBERSHIP_CHANGED_ACTION);
filter.addAction(SimWifiP2pBroadcast.WIFI_P2P_GROUP_OWNERSHIP_CHANGED_ACTION);
SimWifiP2pBroadcastReceiver receiver = new SimWifiP2pBroadcastReceiver(this);
registerReceiver(receiver, filter);
```

Figure 5.2: Code snipe of Android WifiP2p and Termite 1.0 Event Registration.

lems between translating events and explain why translating the information present in the events is also challenging.

One of the main features of Wi-Fi Direct is the possibility to easily establish a connection between two or more devices. Emulate the connection between two mobile devices is complex and requires multiple socket redirection. Termite 1.0 implements socket redirection and encapsulation to provide support to this connection. As seen in Fig.5.6 Termite provides a special socket type that allow the user to deal with sockets as he would normally do in native code. The Wi-Fi Direct Android Framework is different, as we can see in Fig.5.5. The problem is that the conversion from native sockets to termite sockets is made through several indirections. These indirection leads to different behaviours when comparing both socket types. To implement Termite 1.0 sockets the developer would also need to change the application execution flow.

As a conclusion, swapping between Android native code and Termite 1.0 code is not trivial. There are a lot of changes to be made that require application execution flow modifications. These modifications can bring a lot of disadvantages to the developer. It can generate more errors over the ones that he wants to debug. After the Termite 1.0 debug process the developer had two options. He could change the code back to native API, or apply the code modifications to the last working native application code. Both solutions bring huge disadvantages. When the programmer changed the code back to native Android usage, he could generate more bugs, that would need extra effort to debug. Therefore, he would enter a bug loop while swapping between native and termite code. The other solution, to apply the

Android WifiP2p Library

```
@Override
public void onReceive(Context context, Intent intent) {
    ...
    String action = intent.getAction();
    if (WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION.equals(action)) {
        int state = intent.getIntExtra(WifiP2pManager.EXTRA_WIFI_STATE, -1);
        if (state == WifiP2pManager.WIFI_P2P_STATE_ENABLED) {
            // Wifi P2P is enabled
        } else {
            // Wi-Fi P2P is not enabled
        }
    }
    ...
}
```

Figure 5.3: Code snipe of Android BroadcastReceiver.

changes directly in native code also has disadvantages. When the native application code was changed there would have been no way to test if the application was executing properly or not. This would create another test and debug loop between native and Termite 1.0 code.

The solution presented in this thesis brings a lot of advantages when compared to the previous implementation. The ability to test and debug native Android API applications, give the developer the tools he needs to safely deploy his application without the need to further change or re-debug the code. Therefore, Termite is able to recover the normal software development cycle for Encounter Based Network Application developers: Implementation, Test & Debug, Application Deployment. We consider this a major contribution of Termite, as presented in this thesis.

5.3 Emulated Disk Image Performance

Most of the new Termite implementation was made in the Emulated Disk Image. It is essential to have a good performance in the image to provide a good and stable test-bed. Thus, the Termite performance must be close to a real environment. To test the performance of Termite new Emulated Disk Image we used the Wi-Fi Direct application example contained in the Android SDK[4]. This application goal is to share an image through Wi-Fi Direct and it is implemented using Android native Wi-Fi Direct API. To provide a performance measurement, we registered the time spent sending an image through the application. To provide different inputs we repeated the test with 3 different images, which sizes were 5,0 MB, 9,7 MB and 17,6 MB. We ran the test 10 times and registered the results in both the environments, native and Termite. Appendix A of this document shows two of the images used in this test.

As observed in the Table 5.2, Termite presents a huge increase in transfer times. Although Termite presents a higher time to transfer the test image files, we can attribute some of the variation to the emulation process and host operative system. Android Virtual Device present a far worse performance

Termite Library

```
public class SimWifiP2pBroadcastReceiver extends BroadcastReceiver {
    ...
    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        if (SimWifiP2pBroadcast.WIFI_P2P_STATE_CHANGED_ACTION.equals(action)) {
            int state = intent.getIntExtra(SimWifiP2pBroadcast.EXTRA_WIFI_STATE, -
1);
            if (state == SimWifiP2pBroadcast.WIFI_P2P_STATE_ENABLED) {
                    ...
            } else {
                    ...
            }
        } else if (SimWifiP2pBroadcast.WIFI_P2P_PEERS_CHANGED_ACTION.equals(action))
{
            ...
        } else if (SimWifiP2pBroadcast.WIFI_P2P_NETWORK_MEMBERSHIP_CHANGED_ACTION.
              equals(action)) {
             SimWifiP2pInfo ginfo = (SimWifiP2pInfo) intent.getSerializableExtra(
                        SimWifiP2pBroadcast.EXTRA_GROUP_INFO);
            ...
        } else if (SimWifiP2pBroadcast.WIFI_P2P_GROUP_OWNERSHIP_CHANGED_ACTION.
              equals(action)) {
             SimWifiP2pInfo ginfo = (SimWifiP2pInfo) intent.getSerializableExtra(
                        SimWifiP2pBroadcast.EXTRA_GROUP_INFO);
            ...
        }
    }
}
```

Figure 5.4: Code snipe of Termite 1.0 BroadcastReceiver.

Table 5.2: Image send time in milliseconds

		Native Android	Termite
5 MB Image	min	874	6125
	avg	902	8045
	max	946	10032
9.7 MB Image	min	1681	10616
	avg	1788	12036
	max	1910	14650
17,6 MB Image	min	3012	16932
	avg	3164	18496
	max	3343	19438

than real devices. Therefore it is expectable to have a great difference between both results. Intel presented a mobile/emulator performance test[64]. In this test, the real device, when compared to an emulated device running an ARM emulated image, was 10 to 20 times faster. Taking into account those values, Termite results are within a good performance for an emulate environment.

5.4 Summary

Termite was evaluated taking into account three parameters, emulated network deployment, functionality and usability, and Emulated Disk Image performance. The first section proved that Termite behaves as expected in network deployment and can relate the network evaluation provided by Termite 1.0[53]. In the Functionality and Usability Section, we concluded that Termite presents a significant advance in testing and debugging Encounter Based Network applications. This is based on the principle that the developer now does not need to recode the application to test it. In the performance area, Termite per-

Android WiFiP2p Library

```
@Override
protected String doInBackground(Void... params) {
    try {

        /**
         * Create a server socket and wait for client connections. This
         * call blocks until a connection is accepted from a client
         */
        ServerSocket serverSocket = new ServerSocket(8888);
        Socket client = serverSocket.accept();

        /**
         * If this code is reached, a client has connected and transferred data
         * Save the input stream from the client as a JPEG file
         */
        final File f = new File(Environment.getExternalStorageDirectory() + "/"
                + context.getPackageName() + "/wifip2pshared-" + System.currentTimeMillis()
                + ".jpg");

        File dirs = new File(f.getParent());
        if (!dirs.exists())
            dirs.mkdirs();
        f.createNewFile();
        InputStream inputstream = client.getInputStream();
        copyFile(inputstream, new FileOutputStream(f));
        serverSocket.close();
        return f.getAbsolutePath();
    } catch (IOException e) {
        Log.e(WiFiDirectActivity.TAG, e.getMessage());
        return null;
    }
}
```

Figure 5.5: Code snipe of Android socket usage.

formed as expected presenting good results when compared to real environment scenarios.

The next chapter concludes the work present in this Thesis. It also delivers some directions for future work, targeting areas that can extend and improve Termite.

Termite Library

Server

```
private SimWifiP2pSocketServer mSrvSocket = null;
...
try {
       mSrvSocket = new SimWifiP2pSocketServer(10001);
       SimWifiP2pSocket sock = mSrvSocket.accept();
       ...
       sockIn = new BufferedReader(new InputStreamReader(sock.getInputStream()));
       String s = sockIn.readLine();
       ...
} catch (IOException e) {
       e.printStackTrace();
}
```

Client

```
SimWifiP2pSocket mCliSocket = null;
...
try {
       mCliSocket = new SimWifiP2pSocket("192.168.0.2",10001);
} catch (UnknownHostException e) {
       return "Unknown Host:" + e.getMessage();
} catch (IOException e) {
       return "IO error:" + e.getMessage();
}
...
try {
       mCliSocket.getOutputStream().write("Hello World\n");

} catch (IOException e) {
       Log.d("Error reading socket:", e.getMessage());
}
```

Figure 5.6: Code snipe of Termite 1.0 socket usage.

Chapter 6

Conclusions

Encounter Based Networks offer plenty of possibilities to develop applications. While the cloud paradigm is covering most of today's applications, it does not present itself as a solution for every user data exchange challenge. The lack of mobile data and no available Wi-Fi can be an obstacle to the cloud paradigm. There are numerous applications developed for encounter networks, but developers still have a hard time debugging this type of applications.

Application test environments do not support network emulation and network emulation tools do not offer application test and debug tools. Developers could use physical devices, but this method lacks the scalability and automation required. Integrated development and testing environments are required by the new rapid development and continuous integration methodologies, currently used to develop new projects. Termite goal is to cover this gap providing the developers of Encounter based Network applications with an integrated test-bed environment. An example is the Android SDK[4], which support emulated devices, through Android Virtual Devices, but do not provide network emulation for application testing.

In this work we presented Termite. Termite is a distributed test-bed for testing and debugging mobile applications. This project target the Encounter based Networks paradigm, where devices opportunistically form groups, through co-located users that can interact with each other. Termite is a tool that can help developers abstract from the test requirements of Encounter Networks based applications. Through a software emulation service the developer can specify the complexity and dynamically configure an encounter network. We implemented a Termite prototype using Android for the emulated devices, and the Wi-Fi Direct Framework for communication.

In terms of evaluation, we show that Termite implements functionality and usability requirements with good performance. Providing support to Android Wi-Fi Direct Framework API, we support the testing and debugging of encounter based applications without the need to modify the application code. This is important to the usability of Termite, because modifying the application code for testing can bring

disadvantages such as new bugs. Termite presents great advantages in testing applications, since the application is ready to deploy right after testing.

6.1 Future Work

In terms of future work, there are several paths to follow. One of them is the continuous development of Termite focusing cloud deployment. The cloud can be used to deploy virtual images for testing and would greatly improve Termite scalability. That also requires a new interface for node management.

Additionally, the development of an Android Studio plug-in would improve the usability for Termite, making it more user friendly. This would also simplify the Termite deployment in the developers environment, abstracting him from managing Termite Emulated Disk Image or Termite connectors.

Since Termite only supports node displacement through scripting, it would be interesting to support a visual interface for the developer to control nodes. This interface could be integrated in the cloud management interface or in the Android Studio plug-in.

Bibliography

[1] comScore. Number of Mobile-Only Internet Users Now Exceeds Desktop-Only in the U.S. https://www.comscore.com/Insights/Blog/Number-of-Mobile-Only-Internet-Users-Now-Exceeds-Desktop-Only-in-the-U.S, October 2016.

[2] GSMA. Gsma intelligence. https://www.gsmaintelligence.com/. Accessed October 2016.

[3] O. Volovikov, N. Kotilainen, T. Juonoja, M. Vapa, M. Weber, and J. Vuori. Mobile Encounter Networks and Their Applications. *CCNC 2008. 5th IEEE*, pages 1176–1180, 2008.

[4] Google. Android Standard Development Kit. http://developer.android.com/sdk/index.html. Accessed October 2016.

[5] Apple. iOS Standard Development Kit. https://developer.apple.com/library/content/navigation/#section=Platforms&topic=iOS. Accessed October 2016.

[6] K. Y. Jo and S. R. Ali. Design and analysis of large-scale wireless communications networks. *PM2HW2N '06 Proceedings of the ACM international workshop on Performance monitoring, measurement, and evaluation of heterogeneous wireless and wired networks*, pages 18–24, 2006.

[7] S. Basagni, M. Conti, S. Giordano, and I. Stojmenovic. *Mobile Ad Hoc Networking: Cutting Edge Directions*. IEEE Press and Wiley, 2013.

[8] S. Xu and S. Capkun. Distributed and secure bootstrapping of mobile ad hov networks: framework and constructions. *ACM Transactions on Information and System Security (TISSEC) Volume 12, Issue 1, Article No. 2*, pages 1–37, 2008.

[9] C. Zouridaki, B. L. Mark, M. Hejmo, and R. K. Thomas. Robust cooperative trust establishment for MANETs. *SASN '06 Proceedings of the fourth ACM workshop on Security of ad hoc and sensor networks*, pages 23–34, 2006.

[10] Michela Papandrea. Opportunistically supported ubiquitous localization. *MobiOpp '10 Proceedings of the Second International Workshop on Mobile Opportunistic Networking*, pages 211–212, 2010.

[11] M. Conti, F. Delmastro, G. Minutiello, and R. Paris. Experimenting opportunistic networks with WiFi Direct. *IFIP Wireless Days (WD)*, pages 1–6, 2013.

[12] S. Trifunovic, B. Distl, D. Schatzmann, and F. Legendre. WiFi-Opp: Ad-Hoc-less Opportunistic Networking. *CHANTS'11*, pages 1–6, 2011.

[13] M. Pratesi, F. Valentini, and F. Santucci. An Android-based testbed for Hybrid Wireless Mesh Networks in the IoT perspective. *Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI), 2015 IEEE 1st International Forum on*, pages 1–6, 2015.

[14] Vesa A. Korhonen. Mobile Encounter Network - the missing data link. *MindTrek '08 Proceedings of the 12th international conference on Entertainment and media in the ubiquitous era*, pages 80–84, 2008.

[15] J. Kurhinen, V. Korhonen, M. Vapa, and M. Weber. Modelling mobile encounter networks. *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, pages 1–8, 2006.

[16] H. Wirtz, T. Heer, R. Backhaus, and K. Wherle. Establishing mobile ad-hoc networks in 802.11 infrastructure mode. *CHANTS '11 Proceedings of the 6th ACM workshop on Challenged networks*, pages 49–52, 2011.

[17] I. Chatzigiannakis, P. Kokkinos, and C. Zaroliagis. Routing protocols for efficient communication in wireless ad-hoc networks. *PE-WASUN '06 Proceedings of the 3rd ACM international workshop on Performance evaluation of wireless ad hoc, sensor and ubiquitous networks*, pages 90–97, 2006.

[18] C. Schindelhauer, T. Lukovski, S.Rührup, and K. Volbert. Worst case mobility in ad hoc networks. *SPAA '03 Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 230–239, 2003.

[19] The Network Simulator. `http://nsnam.sourceforge.net/wiki/index.php/Main_Page`. Accessed October 2016.

[20] Kevin Fall. Network emulation in the VINT/NS simulator. *Computers and Communications, 1999. Proceedings. IEEE International Symposium on*, pages 244–250, 1999.

[21] A. Kayssi and A. El-Haj-Mahmoud. EmuNET: a real-time network emulator. *SAC '04 Proceedings of the 2004 ACM symposium on Applied computing*, pages 357–362, 2004.

[22] E. Carniani and R. Davoli. The NetWire emulator: a tool for teaching and understanding networks. *ITiCSE '01 Proceedings of the 6th annual conference on Innovation and technology in computer science education*, pages 153–156, 2001.

[23] M. Carson and D. Santay. NIST Net: a Linux-based network emulation tool. *ACM SIGCOMM Computer Communication Review Volume 33 Issue 3*, pages 111–126, 2003.

[24] E. Weingärtner, H. vom Lehn, and K. Wehrle. Device driver-enabled wireless network emulation. *SIMUTools '11 Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, pages 188–197, 2011.

[25] P. Ballal, V. Giordano, P. Dang, S. Gorthi, J. Mireles Jr., and F. Lewis. A LabVIEW based test-bed with off-the-shelf components for research in mobile sensor networks. *Proceedings of the 2006 IEEE International Symposium on Intelligent Control*, pages 112–118, 2006.

[26] K. B. Dhanapal, S. Sankaran, A. A. Somasundara, and S. Pauli. WindTunnel: A Tool for Network Impact Testing of Mobile Applications. *SRII Global Conference (SRII), 2012 Annual*, pages 34–43, 2012.

[27] P. Zheng and L. M. Ni. EMWIN: emulating a mobile wireless network using a wired network. *WOWMOM '02: Proceedings of the 5th ACM international workshop on Wireless mobile multimedia*, pages 64–71, 2002.

[28] K. Ramachandran, S. Kaul, S. Mathur, M. Gruteser, and I. Seskar. Towards Large Scale Mobile Network Emulation Through Spactial Switching on a Wireless Grid. *E-WIND '05: Proceedings of the 2005 ACM SIGCOMM workshop on Experimental approaches to wireless network design and analysis*, pages 46–51, 2005.

[29] R. Beuran, L. T. Nguyen, T. Miyachi, J. Nakata, K. Chinen, Y. Tan, and Y. Shinoda. QOMB: A Wireless Network Emulation Testbed. *Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE*, pages 1–6, 2009.

[30] R. Beuran, L. T. Nguyen, T. Miyachi, J. Nakata, K. Chinen, Y. Tan, and Y. Shinoda. A Multi-purpose Wireless Network Emulator: QOMET. *Proc. of IEEE FINA2008*, pages 223–228, 2008.

[31] J. Nakata, T. Miyachi, R. Beuran, K. Chinen, S. Uda, K. Masui, Y. Tan, and Y. Shinoda. StarBED2: Large scale, realistic and real time testbed for ubiquitous networks. *Proc. of TridentCom 2007*, 2007.

[32] R. Beuran, L. T. Nguyen, T. Miyachi, J. Nakata, K. Chinen, Y. Tan, and Y. Shinoda. QOMB wireless network emulation testbed: evaluation and case study. *WiNTECH '10 Proceedings of the fifth ACM international workshop on Wireless network testbeds, experimental evaluation and characterization*, pages 1–8, 2010.

[33] T. Krop, M. Bredel, M. Hollick, and R. Steinmetz. JiST/MobNet: Combined simulation, emulation and real-world testbed for ad hoc networks. *WinTECH '07 Proceedings of the second ACM international workshop on Wireless network testbeds, experimental evaluation and characterization*, pages 27–34, 2007.

[34] H. Kazemi, G. Hadjichristofi, and L. A. DaSilva. MMAN - A monitor for mobile ad hoc networks: design, implementation, and experimental evaluation. *WiNTECH '08 Proceedings of the third ACM international workshop on Wireless network testbeds, experimental evaluation and characterization*, pages 57–64, 2008.

[35] V. F. Batista and N. S. Rosa. Spontaneousware: a middleware framework for mobile ad hoc networks. *M-PAC '09 Proceedings of the International Workshop on Middleware for Pervasive Mobile and Embedded Computing, Article No. 4*, pages 1–4, 2009.

[36] S. Fujita and H. Esaki. Spontaneousware: a middleware framework for mobile ad hoc networks. *ICUIMC '09 Proceedings of the 3rd International Conference on Ubiquitous Information Management and Communication*, pages 180–184, 2009.

[37] I. M. Lombera, L. E. Moser, P. M. Melliar-Smith, and Yung-Ting Chuang. Peer Management for iTrust over Wi-Fi Direct. *Wireless Personal Multimedia Communications (WPMC), 2013 16th International Symposium on*, pages 1–5, 2013.

[38] Anna-Kaisa Pietiläinen, E. Oliver, J. LeBrun, G. Varghese, and C. Diot. Mobiclique: Middleware for Mobile Social Networking. *WOSN '09 Proceedings of the 2nd ACM workshop on Online social networks*, pages 49–54, 2009.

[39] D. Serfass and K. Yoshigoe. Wireless sensor networks using Android Virtual Devices and Near Field Communication peer-to-peer emulation. *Southeastcon, 2013 Proceedings of IEEE*, pages 1–6, 2012.

[40] H. Zhang, Y. Wang, C. C. Tan, and Y. Zhang. mQual: A Mobile Peer-to-Peer Network Framework Supporting Quality of Service. *2015 IEEE 35th International Conference on Distributed Computing Systems*, pages 754–755, 2015.

[41] Klaus Haller. Mobile Testing. *ACM SIGSOFT Software Engineering Notes archive, Volume 38 Issue 6*, pages 1–8, 2013.

[42] R. Coppola, E. Raffero, and M. Torchiano. Automated mobile ui test fragility: an exploratory assessment study on android. *INTUITEST 2016 Proceedings of the 2nd International Workshop on User Interface Test Automation*, pages 11–20, 2016.

[43] Google. MonkeyRunner. `https://developer.android.com/studio/test/monkeyrunner/index.html`. Accessed October 2016.

[44] Google. Espresso. `https://google.github.io/android-testing-support-library/docs/espresso/`. Accessed October 2016.

[45] PlanetLab. `https://www.planet-lab.org/`. Accessed October 2016.

[46] Emulab - Network Emulation Testbed Home. `http://www.emulab.net`. Accessed October 2016.

[47] J. Bo, L. Xiang, and G. Xiaopeng. MobileTest: A Tool Supporting Automatic Black Box Test for Software on Smart Mobile Devices. *AST '07 Proceedings of the Second International Workshop on Automation of Software Test*, pages 8–8, 2007.

[48] L. Zhifang, L. Bin, and G. Xiaopeng. Test automation on mobile device. *AST '10 Proceedings of the 5th Workshop on Automation of Software Test*, pages 1–7, 2010.

[49] M. C. Calpur and C. Yilmaz. Towards having a cloud of mobile devices specialized for software testing. *MOBILESoft '16 Proceedings of the International Conference on Mobile Software Engineering and Systems*, pages 9–10, 2016.

[50] J. Kaasila, D. Ferreira, V. Kostakos, and T. Ojala. Testdroid: automated remote ui testing on Android. *MUM '12 Proceedings of the 11th International Conference on Mobile and Ubiquitous Multimedia, Article No. 28*, pages 1–4, 2012.

[51] I. Aktas, O. Puña, F. Schimdt, T. Drüner, and K. Wehrle. A framework for remote automation, configuration, and monitoring of real-world experiments. *WiNTECH '14 Proceedings of the 9th ACM international workshop on Wireless network testbeds, experimental evaluation and characterization*, pages 9–16, 2014.

[52] I. Aktas, F. Schimdt, M. Alizai, T. Drüner, and K. Wehrle. Crawler: An Experimentation Platform for System Monitoring and Cross-Layer-Coordination. *IEEE WoWMoM*, pages 1–9, 2012.

[53] R. Bruno, N. Santos, and P. Ferreira. Termite: Emulation Testbed for Encounter Networks. *MOBIQUITOUS'15 proceedings of the 12th EAI International Conference on Mobile and Ubiquitous Systems: Computing*, pages 31–40, 2014.

[54] Openstack. `https://www.openstack.org/`. Accessed October 2016.

[55] Cloudstack. `https://cloudstack.apache.org/`. Accessed October 2016.

[56] Iternational Data Corporation. Mobile devices statistics. `http://www.idc.com/prodserv/smartphone-os-market-share.jsp`. Accessed October 2016.

[57] Google. Android open source projects. `https://source.android.com/`. Accessed October 2016.

[58] Wi-Fi Alliance. Wi-fi direct. `http://www.wi-fi.org/discover-wi-fi/wi-fi-direct`. Accessed October 2016.

[59] Wi-Fi Alliance. Wi-fi certificated devices. `http://www.wi-fi.org/product-finder-results?certifications=34{&}items=3`. Accessed October 2016.

[60] Google. Android open source project, android wifip2pmanager class implementation. `https://android.googlesource.com/platform/frameworks/base/+/refs/heads/master/wifi/java/android/net/wifi/p2p/WifiP2pManager.java`. Accessed October 2016.

[61] Nuno Santos. Termite API. `http://www.gsd.inesc-id.pt/~wiki/courses/cmu1516/lab04/Termite-API.pdf`. Accessed October 2016.

[62] Google. Android Virtual Device. `https://developer.android.com/studio/run/managing-avds.html`. Accessed October 2016.

[63] Google. Android Broadcast. `https://developer.android.com/reference/android/content/BroadcastReceiver.html`. Accessed October 2016.

[64] Intel. Performance results for android emulators. `https://software.intel.com/en-us/android/blogs/2013/12/11/performance-results-for-android-emulators-with-and-without-intel-haxm`. Accessed October 2016.

Appendix A

Test Scripts and Files

In this appendix we present the test scripts and files used to evaluate Termite.

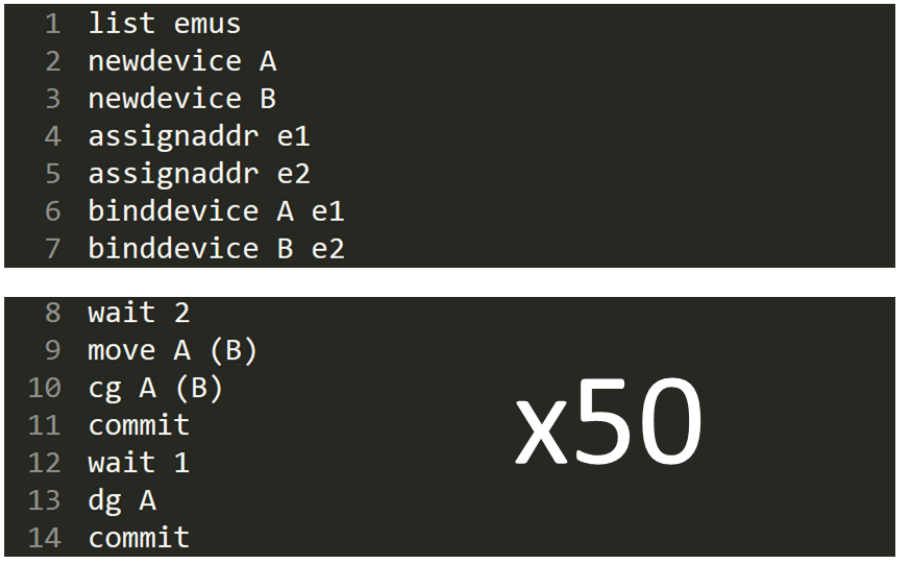


Figure A.1: Termite Script code snipe for the Workload 50 Test.

Figures A.1, A.2 and A.3 are snipes of the Termite Scripts used in Section 1 Chapter 5 of this thesis. Each one reflect different workloads. Figures A.4 and A.5 are the images used in Termite performance test, used in Section 3 of Chapter 5.

```
1  list emus
2  newdevice A
3  newdevice B
4  assignaddr e1
5  assignaddr e2
6  binddevice A e1
7  binddevice B e2
```

```
8  wait 2
9  move A (B)
10 cg A (B)
11 commit
12 wait 1
13 dg A
14 commit
```

x100

Figure A.2: Termite Script code snipe for the Workload 100 Test.

```
1  list emus
2  newdevice A
3  newdevice B
4  assignaddr e1
5  assignaddr e2
6  binddevice A e1
7  binddevice B e2
```

```
8  wait 2
9  move A (B)
10 cg A (B)
11 commit
12 wait 1
13 dg A
14 commit
```

x250

Figure A.3: Termite Script code snipe for the Workload 250 Test.

Figure A.4: 5MB Image used for Termite Performance Test.

Figure A.5: 9,7MB Image used for Termite Performance Test.

Appendix B

Performance Test Results

In this appendix we present, in full extent, the test results set demonstrated in Termite evaluation. Table B.1 shows the entire results presented in Section 1 of Chapter 5 and Table B.2 shows the entire results presented in Section 3 of Chapter 5.

Table B.1: Workload execution times in seconds

	Termite 1.0	Termite
Workload 50	160,89	160,80
	158,93	160,87
	159,22	157,07
	162,13	156,85
	159,76	157,16
	159,29	156,67
	160,56	155,76
	160,02	157,52
	159,12	157,11
	160,01	156,29
Workload 100	315,59	310,14
	315,66	309,91
	312,33	309,85
	313,34	310,85
	312,48	309,74
	315,01	310,09
	313,98	309,10
	314,81	309,80
	314,04	310,71
	315,22	309,90
Workload 250	773,19	770,13
	773,13	771,81
	772,84	771,04
	773,14	770,55
	773,89	771,01
	774,08	771,80
	773,11	770,93
	772,98	770,56
	773,54	770,82
	772,81	771,02

Table B.2: Image send time in milliseconds

	Native Android	Termite
5MB Image	881	7112
	938	8554
	946	8178
	895	6914
	875	6125
	901	8901
	890	10032
	874	7891
	915	8445
	908	8298
9,7MB Image	1712	11327
	1753	11032
	1665	11009
	1910	11432
	1832	13455
	1784	12855
	1760	10616
	1681	13258
	1903	14650
	1884	10826
17,4MB Image	3087	18680
	3199	18665
	3107	18988
	3223	16932
	3143	18522
	3228	19438
	3012	18330
	3343	18823
	3211	17947
	3095	18635