Termite: Emulation Test-Bed for Encounter Networks

Filipe Teixeira
filipelteixeira@gmail.com

Instituto Superior Técnico, Lisboa, Portugal

October 2016

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:** mobile devices, encounter based networks, emulation test-bed, android, wi-fi direct

1. Introduction

In the last years, mobile devices became the primary computational platform for users[10]. With officially more devices than people in the world[17], 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[23].

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.

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. 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.

This scenario 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.

1.1. 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. 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 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, and **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.

1.2. Current Solutions

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[14, 4]. 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.

1.3. Contributions

This work provides the following contributions: (i) architectural design of Termite test bed as well as the interactions, protocols and procedures, (ii) architectural design of the Emulation Image used for the mobile devices in the Termite system, (iii) implementation of new connector between the Termite and the Mobile Emulated devices, (iv) implementation of the Mobile Emulation Image augmentation in an Android prototype, (v) evaluation of the emulation effects in the Encounter Based Network environment, and (vi) evaluation of Termite performance impact comparing to real world scenarios.

1.4. Document Structure

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

2. Related Work

In this section we discuss and compare several systems to Termite.

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.[19] compare several networks, relating wireless networks

with mobile ad-hoc networks.
Mobile Ad-Hoc Networks or simply MANETs[5] are a well researched area but still with some pitfalls. One of them is the secure MANETs bootstrapping. Xu et al.[24] 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.[26] 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.
The closest description to Encounter Based Networks is made by Korhonen[21] and first quoted by Kurhinen et al.[22]. 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 [23] 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

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.
Some of these solutions only provide network simulation, like NS[2], while others, like EmuNET[20] and NIST Net[9], provide network emulation. Other systems, as NetWire[8], increase the scalability supporting network emulation over an internet connection. This system shares many of the same issues. The lack of user displacement or application testing support makes testing a difficult task for the programmer. QOMB[6] is a test bed designed and implemented for the evaluation of wireless network systems, protocols and applications. But QOMB does not support mobile application development because it lacks the appropriated automated testing tools and it is bounded to one infrastructure, which 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 a crescent 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 for 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.
ATMOS[12] 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 well in Encounter Based Network paradigm. Another important system is mQual[25]. mQual is a mobile peer-to-peer network framework, implemented as an upgrade to the Wifi-Direct framework. With mQual being essentially a performance upgrade to the Wifi-Direct protocol, it has the same drawbacks found in ATMOS, lack of network emulation and application automation test.

2.4. Automated Test Frameworks

Application test automation has been a point of interest since the advent of application development. Unlike other programs that can be tested through test battery, applications add the UI factor to the equation. UI test automation tools have presented huge improvements. However simple application testing already have enough fragilities[11], encounter based application add the distributed factor issue. Termite needs to provide application test support to the developer.
MonkeyRunner[16] and Espresso[15] are simple tools that 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 of coordinate both tools would be higher than build the entire system from scratch.
PlanetLab[3] and EmuLab[1] are frameworks ready for emulation tests. These frameworks have the capability to aid 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.

2.5. Termite Background

Termite 1.0[7] 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 identify the main reasons why is Termite a necessary upgrade to Termite 1.0.

Although Termite 1.0 is the background of Termite, we there are some requirements described in the first section that are not fulfilled by Termite 1.0. The two main unfitted requirements are flexibility and usability which both are very important for the user experience. Termite 1.0 do not support the usage of physical devices in the emulation process. It also do 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 allow developers to test their applications using the native code.

3. Architecture

This section presents the Termite architecture design. This architecture is based on the Termite 1.0[7]. Termite extends Termite 1.0 architecture and improves it to fulfil the requirements presented, adding new components and functionalities to the system.

3.1. Overview

Termite supports two front-ends and multiple back-ends. 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. Devel 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 front-ends are dynamically configurable to use the back-end 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.

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.

The Emulated Disk Image is the solution to fulfil the Termite requirements and solve Termite 1.0 issues. In Termite 1.0 the emulated image is enforced with extensions that implement the components responsible for Termite 1.0 execution. 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 introduce a new component, the Termite Controller. 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.

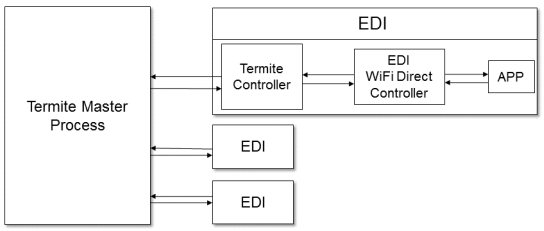

Figure 1: Interaction between the Master Process and the Controller inside the Emulated Disk Image

The interactions between the Controller and the Master Process remain the same, as well as the ones between the Emulated Disk Image and Termite. The main changes are explained by describing the Emulated Disk Image architecture and how the Termite components are added to it, as shown in Fig.1.

3.2. 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. 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.

3.3. Controller Event Management

As explained previously, 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 it is not trivial, because they are not just new events, but they carry information about the emulated network, or the emulated image.
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 also EDI specific. 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.
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 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 take us to the next topic, how does the Master Process relates with the rest of the system.

3.4. Termite Master Process - EDI Interaction

The Master Process is a core component of the system with several responsibilities key for the overall system execution. One of them is the interaction with the Termite Front-end. The Master Process receives the developer settings and transform 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 lead to event triggering.
The abstraction between the Master Process and the EDIs is critical for the system. Adding the abstraction layer improves Termite 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 present similar behaviour while the application is running. The second requirement is more complex than the first. When the same application interact with different Clients but using the same input it must be expectable that, under the same conditions, the output would equal. Regarding the platform running the EDI, there must be a connector that ensure correct interaction with the Termite system. This connector was addressed on previous section.
MP plays an important role in creating the connection between EDIs. The events triggered by the MP contain the information necessary for the EDIs to create channels between them. After receiving the Connection event, the EDI chosen by the developer will connect to the the second EDI setting up a communication channel.
After both EDIs are connected, the Master Process is not responsible any more for the interaction between them. The MP only interacts again with either one of them in two cases. First, a 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. Here the Master Process receives the command and send the correct event to one or both of them.
After the EDIs being connected, it is time to exchange information among them. The application has the freedom to chose alternative message for-

warding mechanisms which are not 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.
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.

4. Implementation

In this section we present the prototype design and explain the major challenges found in the implementation. The Termite prototype was developed on top of Android Marshmallow(6.0.1).

4.1. 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.

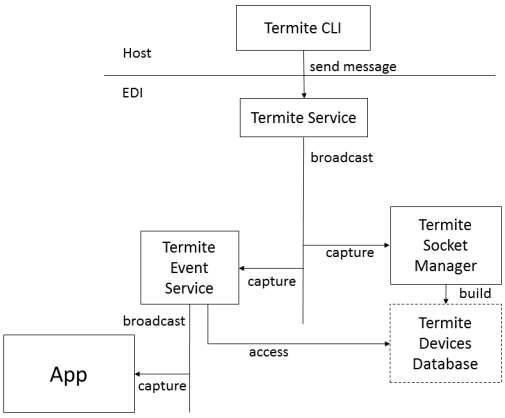

Figure 2: Termite Implementation Design

In Fig.2 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 translate the Termite Event to a WiFiP2p 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 access the database that the Termite Socket Manager builds. When the WifiP2p event is broadcasted, the application captures it and proceeds with its normal execution.

4.2. Emulation Controller - Broadcast Flow Control

As seen in Fig.2, 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[13] 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. 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.
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. 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 implementation or Termite. 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 by 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. The switch is implemented inside the menu Settings >Wireless & Networks >More. This allow us to register all the needed broadcast receivers and also allow us to unregister them when the emulation is turned off. Termite Enabler is a Emulation Controller component which is responsible for controlling the broadcast receivers service.

4.3. 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 the conversion 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 the virtual devices, and CV addresses are used locally to represent the emulated image.

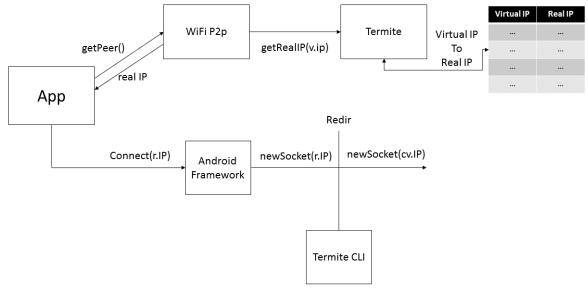

Figure 3: Socket Redirection in Termite

In Fig.3 we can observe that Termite is condensed to one single component. The unique goal of Termite interaction in this phase is to convert the virtual address to a real one. The stock Android Wi-Fi Direct implementation expect 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.

Table 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

5. Evaluation

In this section, we present the evaluation and corresponding results collected throughout this work. For the tests presented in this section, 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 the 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.
Although, to prove the idea that Termite network management relates 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, if not identical, to Termite 1.0. To achieve such conclusion, the test is composed by successive creation and deletion of a network between two nodes in a local machine. 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 1, Termite has a performance not worst then Termite 1.0. In all the workloads Termite performance matches Termite 1.0, having always 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.
We can conclude that in terms of network deployment and management, Termite performs well and reflect the evaluation presented in Termite 1.0.

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. 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 is. This is a great improvement that bring a lot of advantages to the developer. The need to rewrite the code to use a different API is no longer an obstacle to application test, as it was in Termite 1.0. In this section we will address the changes between application code that grant Termite improved functionality and usability when compared with Termite 1.0. We will also describe the advantages of using native code during the test and debug process, as well as the application market deployment steps that are significantly improved.
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. And after the Termite 1.0 debug process the developer had two options. He could changed the code back to native API, or apply the code modifications to the last working native application code. Both of these 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. The other solution, to apply the changes directly in native code also has disadvantages. When the native application code was change, there would be no way to test if the application is now executing properly or not. This would create another test and debug loop between native and Termite 1.0 code.
The solution presented in this paper brings a lot of advantages when compared to the previous implementation. To be able to test and debug native Android API applications, give the developer the tools he need 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.

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 Image Disk we used the Wi-Fi Direct application example present in the Android SDK. 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 that took to send 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.

Table 2: Image send time in milliseconds

		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

As observed in the Table 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 than real devices. Therefore it is expectable to have a great difference between both results. Intel presented a mobile/emulator performance test[18]. 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.

6. Conclusions

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

presents good performance, functionality and usability. By 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 present great advantages in application testing, since the application is ready for deploy right after testing.

References

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

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

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

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

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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. Accessed October 2016.

[11] 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.

[12] 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.

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

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

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

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

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

[18] 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.

[19] 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.

[20] 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.

[21] V. 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.