Termite2 - Supporting Scalable and Usable Encounter-based Apps

Fernando Daniel Alves Moreira

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors: Prof. Paulo Jorge Pires Ferreira
Dr. Rodrigo Fraga Barcelos Paulus Bruno

Examination Committee

Chairperson: Prof. David Manuel Martins de Matos
Supervisor: Prof. Paulo Jorge Pires Ferreira
Member of the Committee: Prof. João Coelho Garcia

January 2021

Acknowledgments

First of all, I would like to thank my parents and my brother, for their love, encouragement and patience over all these years, for always being there for me through thick and thin and without whom this project would not be possible. To my girlfriend Mariana, thank you for all the love, help and support, you made this journey much easier.

I would also like to acknowledge my dissertation supervisors Prof. Paulo Ferreira and Dr. Rodrigo Bruno for their insight, support and sharing of knowledge that has made this thesis possible.

Lastly, I would like to thank all my friends and colleagues that helped me grow as a person and were always there for me during the good and bad times in my life. Thank you.

To each and every one of you – Thank you.

Abstract

Today, almost every mobile application requires some form of communication technology. This is specially true for data-sharing applications. These applications normally provide sharing functionalities mostly through the internet, even when users are in proximity to exchange data via peer-to-peer networks. This shows a lack of peer group solutions within data-sharing applications. To solve this problem a new paradigm and solution has emerged, the Encounter Networks paradigm. Unfortunately, proper support for developing and testing applications that apply this network paradigm is still lacking. Without the ability to easily develop and properly test applications that support encounter networks, developers are forced to publish applications without proper testing or to choose more traditional forms of data sharing and communication for their apps, i.e., through the Internet. In this project we present Termite2 the next version of the system Termite, with improved system scalability and usability. Termite2 provides an emulation test-bed solution for encounter network applications, allowing the user to create/model encounter networks in a dynamic way to translate user interactions using Android virtual devices. Currently, Termite has limited scalability and usability, not providing proper support to create large emulated networks with a larger number of virtual devices. Termite2 improves Termite's scalability by allowing emulators to run distributed across multiple local or remote machines and it improves usability by providing a new graphical user interface option from where the emulated network is created and modeled. Termite2 was implemented using Java and the new graphical user interface using JavaScript which integrates nicely with the Google Maps API to display an interactive map of the emulated network. Termite2 can run on Windows, Linux or Mac and supports Android mobile applications and virtual devices.

Keywords

Termite2; Termite2 Client; Termite2 Server; Termite2 API; Network emulation; Encounter Networks; Android Development; Wi-Fi Direct; Android virtual devices; Emulators;

Resumo

Hoje em dia, quase todas as aplicações móveis requerem algum tipo de tecnologia de comunicação, tornando-se especialmente importante para aplicações onde a partilha de dados é necessária. Estas aplicações geralmente fornecem estas funcionalidades de partilha de dados utilizando o acesso à Internet, mesmo quando os utilizadores se encontram em proximidade para realizar troca de dados utilizando redes *peer-to-peer*. Este facto mostra uma falta de soluções para aplicações móveis que utilizem tecnologias de comunicação *peer-to-peer* para partilha de dados. Acontece também que devido a uma falta de ferramentas de desenvolvimento que facilitem e ajudem na criação e teste de aplicações que usem este tipo de comunicações, programadores são obrigados a publicar aplicações sem realizar os testes apropriados ou então a escolher métodos de comunicação mais tradicionais para as suas aplicações, tal como, comunicação via Internet. Neste projecto apresentamos o Termite2, uma evolução do antigo sistema Termite, com elevadas melhorias de escalabilidade e usabilidade do sistema. Tal como o Termite, o Termite2 é uma ferramenta de emulação de redes de encontros para o desenvolvimento e teste de aplicações que utilizem redes de encontros como paradigma de comunicação. O sistema permite criar e modelar redes de encontros de forma dinâmica onde interações entre os vários elementos da rede são traduzidas para comunicações entre dispositivos móveis virtuais. Infelizmente a escalabilidade e usabilidade de sistema do Termite é bastante limitada. O Termite não permite ao utilizador desenvolver testes sobre uma rede virtual de grandes dimensões com um elevado número de dispositivos móveis virtuais. O Termite2 soluciona o problema da usabilidade permitindo a criação e modulação das redes virtuais através de uma interface gráfica e melhora a escalabilidade do sistema permitindo destribuir os dispositivos móveis virtuais ao longo de várias máquinas locais ou remotas. O Termite2 foi desenvolvido utilizando Java e a nova interface utilizando JavaScript e a Google Maps API. O sistema pode ser utilizado em Windows, Linus ou Mac e suporta o uso de aplicações e emuladores Android.

Palavras Chave

Termite2; Termite2 Client; Termite2 Server; Termite2 API; Emulação de rede; Redes de Encontros; Desenvolvimento Android; Wi-Fi Direct; Dispositivos virtuais Android; Emuladores;

Contents

List of Figures

List of Tables

Acronyms

GUI	Graphical User Interface
OS	Operating System
SNT	Scalable Network Technologies
NOEL	Network Oriented Emulation Language
AVD	Android Virtual Device
adb	Android Debug Bridge
RNL	Rede das “Novas” Licenciaturas
IST	Instituto Superior Técnico
UML	Unified Modeling Language
VNC	Virtual Network Computing

1 Introduction

Contents

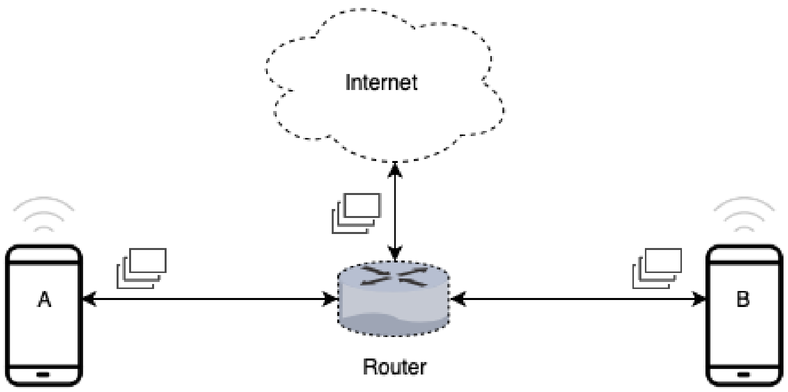


Figure 1.1: Communication between two devices via the Internet

With more mobile devices than people in the world [1] and with their ever growing numbers and computational power, mobile devices have become the primary computational platform for users [2] and are today an invaluable tool in every day life.

Today, almost every mobile application requires some form of communication technology and this is specially true for data-sharing applications. These applications normally provide sharing functionalities through the internet, using Wi-Fi or broadband cellular network. For example, suppose two users want to share some file while being in the same physical location; to share this file, most data-sharing applications establish a connection to a remote central server or some kind of redirection service to exchange the data between the users, even though they are co-located. This example shows a lack of peer-to-peer and peer group solutions within data-sharing applications. Thus, when mobile devices are co-located, a new paradigm is now possible, the Encounter Networks paradigm [3].

The term encounter networks represents a special case of **ad-hoc networks** that targets such mobile devices; when these devices are co-located and involved in social interactions, they make an encounter network.

1.1 Motivation

Encounter Networks provide an important shift regarding device to device communication and data sharing. Instead of relying on a central access point (router) with an Internet connection to establish communication and data transferring (see Fig. Figure 1.1), mobile devices can now use peer-to-peer communication technologies like Bluetooth or 802.11 WLAN to achieve the same results when devices are near each other. Another example of this communication solution occurs when co-located devices use Wi-Fi Direct [4]. Wi-Fi Direct is a type of encounter network technology that provides communication between devices based on their location and proximity, without the need for a central access point or internet connection [5, 6].

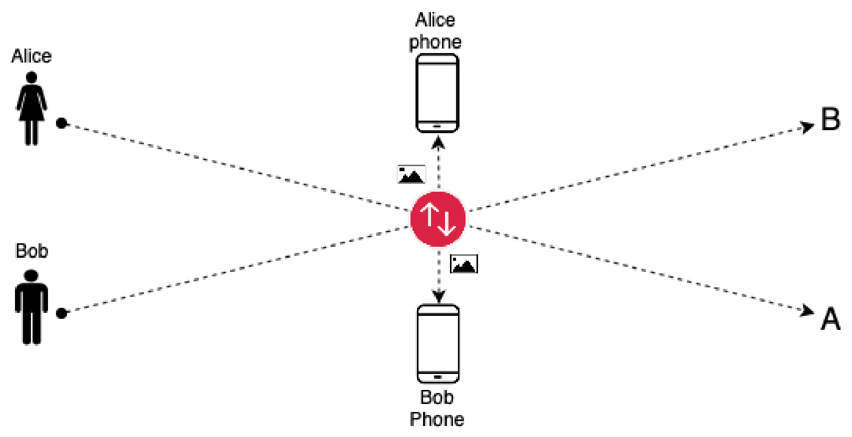


Figure 1.2: Alice app in action

Let us then consider a scenario where a mobile developer, Alice, wishes to create a photo sharing application for Android devices where users can automatically share photos between them when near each other. Assuming that Alice wants to support an encounter network, she may choose an already available communication technology for this paradigm, Wi-Fi Direct, as the communication technology for her application. Fig. 1.2 illustrates Alice and Bob moving to point A and B respectively; the red mark where both paths cross, represents the point where Alice and Bob come in range for Alice app to start automatically sharing photos via WiFi-Direct.

When developing her app, scalability is one of Alice application requisites. Her application should be equally capable of handling a small or large number of users connected at the same time. With currently available solutions, Alice would have to gather dozens of Android devices in order to accurately test the app (using Wi-Fi Direct communication), while also simulating reliable device displacement to simulate users coming and going out of range between each other. Alice can not afford this type of scalability tests for several reasons (cost, time, complexity). Therefore, Alice is forced to publish her application without proper testing or instead choose another communication technology for her application. This scenario clearly shows a problem regarding the inability to easily and properly develop and test applications that support encounter networks.

Android provides its own development tool kit with support for virtual emulation of android devices and frameworks for automated application testing called Android Studio [7]. However, this tool does not provide the necessary support to develop and test applications that apply the *encounter network* paradigm. The Android Standard Development Kit does not implement multi-node emulation and displacement using the emulated android devices, which is required to properly test encounter network scenarios. And our related work Section 2 shows that the network simulation and emulation tools available today, while able to offer the necessary network layer capable of simulating/emulating encounter networks, offer no support for testing encounter-based applications on top of the network created. While, tools specially designed for application testing lack the necessary network layer.

A solution to this lack of support would be to use the Termite system [5] [8]. Termite is a test-bed solution that allows developers to create and model encounters network where android devices can be used to test encounter based application. However, as we will discuss on our related work section, Termite system shows a level of system usability and scalability that is not satisfactory when we consider the nature of the applications that it helps develop and test.

1.2 Goals

The main goal of this project is to develop Termite2, an evolution of the previous system Termite, offering improved scalability and usability. Termite2 must be capable of creating both small and large emulated encounter networks and support the development and testing of Android mobile applications that use the encounter network paradigm.

In a typical development cycle, the developer would first start by designing and coding her/his application. The following development phases would be supported with the usage of Termite2. Thus, Termite2 allows the developer to build an encounter network in order to properly emulate a real world encounter network. Termite2 achieves this by providing the ability for network nodes (where each network node represents an Android virtual device) to move and interact based on proximity, thus mimicking real world mobile encounters. After creating such network, Termite2 allows the developer to deploy the application on virtual network nodes and run automated tests. These tests are crucial to help developers understand their application behaviour on a real world scenario of an encounter network. As previously mentioned, Termite2 must be able to properly emulate encounter networks. This includes both, small and large networks in order to provide a wide range of real world scenarios, where developers can accurately test their applications. Termite2 also supports the ability to run with a step-by-step execution, allowing the developer to inspect the execution state of each network node.

Termite2 has four main components. The first is the Termite2 Client that runs locally on the developer machine, and is inside this component that the emulated network is created/modeled. On the emulated network virtual nodes can be bound to Android virtual devices which then allow us to emulate peer-to-peer interaction between them.

The second component is the Termite2 Server. The Termite2 Server runs on any machine that we wish to run emulators on to use on the emulated network. The Termite2 Server is responsible for managing the Android emulator instances that run on the same machine as the Termite2 Server. The Termite2 Client is then able to connect to multiple Termite2 Server(s) and use the emulators managed by the them on the emulated network.

The third components is the Termite2 API. The Termite2 API is an Android library that must be used by the applications that we wish to test/develop using the Termite2 system.

Finally, the fourth component is Termite2's new Graphical User Interface (GUI), called Termite2 GUI. The Termite2 GUI is accessed from a web-browser and allows the user to interact with the Termite2 system through interactive clicks and selections. Using this GUI, the user is able to see, create and model the emulated network on a real world map, emulate complex peer-to-peer scenarios, and manage multiple emulators distributed across multiple machines.

As such, when we consider the previously presented system goal and Termite2's components, the Termite2 system should fulfill the following system requirements:

Usability: Termite2 must help developers creating and testing their mobile encounter applications in a quick and easy way. It must have a clear and useful GUI in order to allow for developers to easily see and model their emulated networks and visualize virtual nodes movement and interactions. This visual interface must be displayed on a real world map in order to better mimic real life scenarios on a real world environment. From this interface the user must also be able to manage the necessary Android emulators.

Performance: The overall system must have good performance. Termite2 must be similar or better then Termite, when we consider the emulated network and tests that can be produced on both systems.

Scalability: Termite2 must be able to create and handle small and large emulated networks. To achieve it Termite2 must be able to support/manage both small and large numbers of virtual Android devices. Deploying small or large numbers of Android emulators should be easy and fast.

Flexibility: Termite2 must be able to run on common Operating System (OS) like Windows, Linux and Mac. It must support emulated Android devices which can run locally (i.e., on the same machine as Termite2 Client) or distributed through multiple machines.

1.3 Contributions

The contributions of this project are seen through the new additions and improvements done on the old Termite system, which unlock core issues in Termite, namely usability and scalability. Termite2 is a new solution for the problem of creating and managing all aspects of Android emulated devices running locally or remotely without the need to use Android Studio or custom made user scripts. As a matter of fact, to the best of our knowledge, there is no other system such as Termite2. Therefore, this project contributions are the following:

Distributed architecture: Termite2's supports a distributed approach to manage the emulated network and the virtual Android devices. The emulated network is created/modeled on the Termite2 Client and virtual Android devices are managed by the Termite2 Server(s). Termite2 Server was designed to work with the Termite2 Client, but can also be used as a stand alone system to create and manage small and large numbers of emulated Android devices distributed across multiple machines from a single control point.

System scalabilty: Termite2's new distributed architecture allows a user to distribute the computational load of running multiple Android emulators throughout different machines. The size of the emulated network is no longer constrained neither by the number of emulated devices that the user can run on the same machine where the emulated network is created nor the limit imposed by Android Studio. This system architecture, when compared to the previous Termite system, greatly improves the number of virtual devices that the user can use; it allows for the creation of much larger emulated networks that can better emulate real world scenarios. Consequently, Termite2 gives the user the ability to create better and more extensive tests for their mobile applications.

Termite2 new interface: We built a new GUI using JavaScript and the Google Maps API. This new interface is accessed from a web-page and runs on the Apache Tomcat Server. This new interface allows the user to create, manage and visualize the created emulated network and all virtual device interactions on Google Maps. Proximity between the network virtual nodes and WiFi-Direct Peer-to-Peer groups creation is now done based on the emulated coordinates of each virtual node (on Google Maps). Node interactions (ex: node A coming in to proximity of node B) can be done manually or automatically moving the nodes along defined paths. Android emulators running distributed across multiple machines can be easily managed and bound to each network node using this interface.

System usability: Termite2 provides multiple usability improvements. The major one comes from the new GUI. Termite2's new interface allows the user to see the emulated network and all the interaction that can happen within. User interaction with the new interface is done through interactive click and selections instead of written commands on a terminal windows (like it was the case when using the old Termite version). Creating and managing Android emulators is now much easier and can be done from the system itself, without the need to use Android studio or user made script to perform these tasks like it was when using Termite. We are now able to run emulators across multiple machines and managed them from a single control point, the Termite2 Client. Finally, Termite2 also simplifies and automates some configuration aspects of the system. For example, in order to use the emulators on the emulated network some port redirection rules have to be set on them. On the Termite system these rules are set manually by the user, while on the Termite2 system these rules are set automatically (for a more detailed explanation please see Section 3).

1.4 Document structure

This project's document is organized into six sections. Following the Introduction, we have Section 2, where we describe and discuss a number of different systems that can be related to Termite2 main goals and requirements. We explore what each system was design to do in order to see if the system presents a valid solution to the problems we described on Section 1.2; On Section 3 we explain in detail our project architectural basis and our architectural solution, a new version of Termite system called Termite2. On Section 4 we present and discuss the most important aspects of Termite2 implementation. Section 5 covers the evaluation performed on Termite2 system, which includes usability, performance and scalability tests. The test were performed in order to evaluate Termite2 system against the goals and requirements presented on Section 1.2. Finally, we have Section 6 with the conclusion and future work.

2

Related Work

Contents

This chapter is organized in six Sections: 2.1 Encounter Networks, 2.2 Network Simulation, 2.3 Network Emulation, 2.4 Test Frameworks, 2.5 Termite and 2.6 Discussion.

If we simplify Termite2 to its core functionalities, we can represent the system as a tool that provides the ability for developers to execute automated mobile tests running on Android emulators on top of a network layer, properly modeled to express Encounter Networks and all its characteristics (see Fig. 2.1). Therefore each of the following sections of related work focus on a particular layer of this systems and the requirements and goals set for this project on Section 1.2. Another important consideration to take before delving on this chapter is that, although Termite2 presents itself as a evolution of the previous system Termite, more than five years have passed since the creation and presentation of Termite [5], and a new analyse of the currently available tools and solutions to the problems presented on chapter 1 is required, including the Termite system itself.

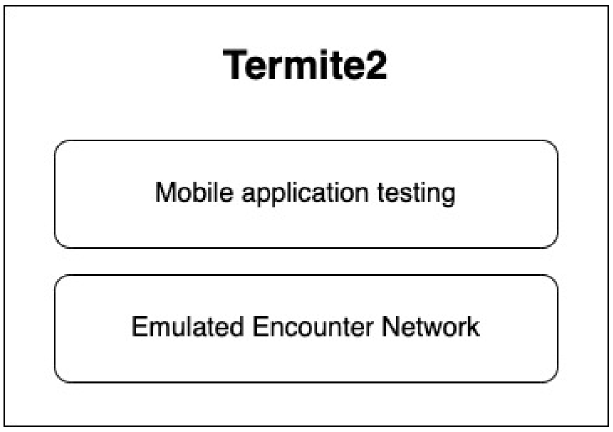


Figure 2.1: Simplified view of Termite2 system layers

First, in Section 2.1 (Encounter Networks), we start by showing and discussing other network paradigms than the one considered on this project, Encounter Networks. We then target the analysis of current solutions that can be related to Termite2 simplified network layer. In Sections 2.2 (Network Simulation) and 2.3 (Network Emulation), we discuss tools that can simulate or emulate networks respectively, distinguishing the two different approaches as network tools and see if they can help in the development of mobile application that target encounter networks by providing mobile application testing support (application must run from emulated mobile devices) on top of the created network. Next we discuss the current systems that fit the simplified Termite2 testing layer. In Section 2.4 (Test Frameworks), we analyse popular test frameworks used in the development of mobile application, to see if the available tools provide the proper network layer support needed in the development of mobile application that use encounter networks. Section 2.5 (Termite), is centered on the previous version of our project system, Termite. We will analyze the current Termite solution to see where the system succeeds and fails when considering the Termite2 requirements and goals.

Finally, in Section 2.6 (Discussion) we will summarize the most important aspects discussed on this chapter and present the basis for our current solution.

2.1 Encounter Networks

We have previously stated that encounter networks [3] can be seen as a special case of ad-hoc networks. Both concepts apply the notion of decentralized wireless networks where the network is created without the need of a preexisting infrastructure, such as routers or access points to discover and establish communication, but encounter networks exclusively target social interactions amongst co-located mobile devices. Nevertheless, encounter networks can be compared to other network definitions that also explore and apply the concept of ad-hoc network, such as mobile ad-hoc network.

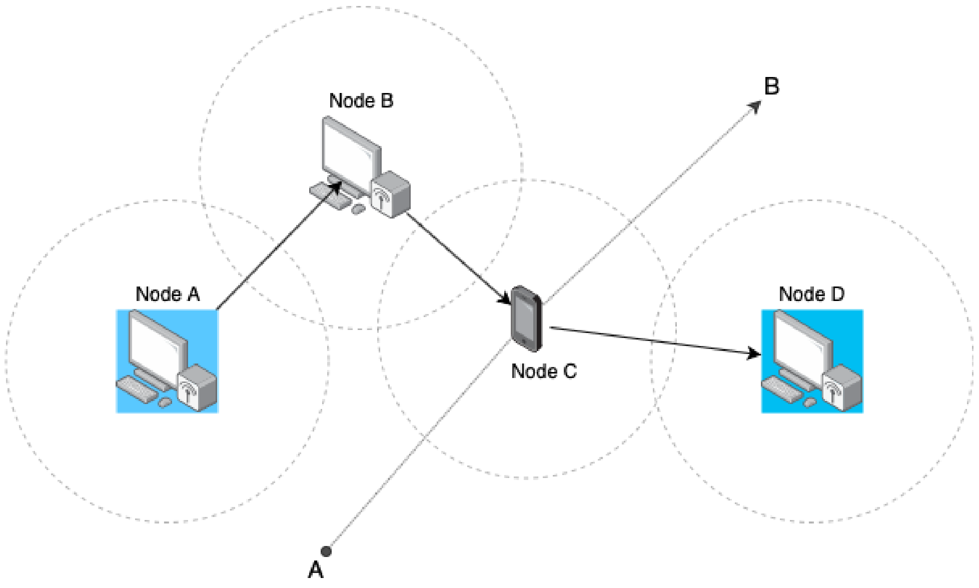


Figure 2.2: Communication within a MANET network

In mobile ad-hoc networks or MANETs [9], network nodes can represent static or mobile devices (for example a computer inside a building or smartphones) and all nodes inside the network can communicate with each other, even if they are not in range of a direct wireless signal. This direct communication is achieved using multi-hop communication (nodes forward each other's packets), where a node can communicate with other even if they are not in range by using other nodes as communication bridges between them (see Fig. 2.2, where node A can communicate with node D, even if out of wireless range, by using node B and C as communication hops between them. Nodes can be mobile within the network as shown with node C, moving from point A to B).

In Encounter Networks, nodes are dynamic and represent mobile devices (for example a smartphone or a tablet) and nodes can establish communication encounters with multiple nodes but, only when in range. Support for multi-hop communication is not present. This is because the time of interactions between two devices is normally short and device density in a given area can be too low to provide good multi-hop capabilities or too high causing the routing protocol to take too much time and consume too much energy, and these are fundamental factors to consider when talking about mobile devices, where energy and computational power are very limited. Although some work has been produced in order to directly reduce or provide a solution to this problems, for example increased routing protocol efficiency [10] in order to reduce energy consumption and power computation, the results do not show clear improvements or present unfeasible approaches [11] that use fixed infrastructures as network support.

Mesh, Sensor, Opportunistic and Vehicular Networks have emerged from the MANET world as a more pragmatic application of the multi-hop ad hoc networking paradigm [12]. But all of this specializations suffer from the same problems as the ones present in MANETs.

2.2 Network Simulation

Network simulators are software solutions that can perform tasks in abstract to demonstrate the behavior of a network and its components, without performing the real and concrete behaviour of these components or networks (simulation normally uses mathematical formulas to mimic a specific component operation). Network simulators are normally used primarily for research and educational purposes as testing tools in the design and development of a network itself. This tools may appear as an obvious choice to properly recreate encounter networks and all its characteristic. Unfortunately the concept of Network Simulation itself presents a major flaw when we consider this project scope. To properly present this problem let us discuss some popular network simulators.

NS-2 [13] is an open source, object oriented TCL (OTcl [14]) script interpreter with network simulation event scheduler. It provides unique network component object libraries and network setup module libraries. This network simulator is used for setting up and running wired or wireless network simulations that a user can use to perform various network tests. In order to do this, the user can write a simulation program in OTcl script language, to initiate the event scheduler, setup the network topology and tell the traffic source when to start and stop sending packets through the event scheduler. NS-2 can be used to extensively test new protocol solutions for various network paradigms including MANETs and encounter network. After modeling the desired network topology, it is possible to execute various network operations in order to test and study how the network behaves under specific constrains. NS-2

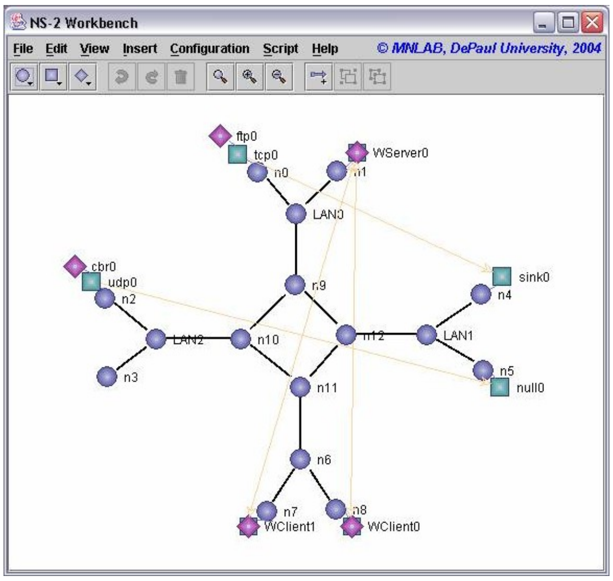


Figure 2.3: NS-2 Graphical User Interface

is feature-rich when considering protocol and network testing, but shows poor scalability, regarding the number of network elements (nodes) that a network can have when using this tool. NS-2 also provides various graphical user interface options that allow the user to see the network and all node interactions (see Fig. 2.3, that shows an example of a network created with NS-2 with various elements and their connections), providing detailed information about the network state and connections. Sadly, this user interface is custom made to work with NS-2 OTcl scripts, making the idea of adapting this interface for our project too time consuming and difficult. Nevertheless the real problem that renders this tool incapable of supporting our project is the fact that NS-2 has no support for mobile application testing, since it does not support mobile virtual devices as network nodes of the simulated network.

NS-3 [15] was developed in order to improve upon the core architecture, software integration, models, and educational components of NS-2, while maintaining almost all features. The major improvement over NS-2 is that instead of relying on OTcl as its scripting environment, NS-3 uses C++ programs or python scripts to define the simulations. This made the tool significantly easier to use and build on top of. Unfortunately, NS-3 still has the same problems as the ones identified on NS-2; poor system scalability and no support for mobile application testing on virtual devices running on top of the simulated network, thus making this tool not suitable for our project needs.

GloMoSim [16] is a parallel discrete event network simulation tool. It gives the ability to simulate thousands of nodes (executing in parallel or in different machines) which can be linked by diverse communication capabilities such as asymmetric communication, multi-hop wireless communication using Ad-Hoc networking or more traditional Internet Protocols. GloMoSim is built as a set of libraries and is especially oriented to simulate wireless ad-hoc networks, which includes encounter networks. Glo-

MoSim libraries are built in Parsec (a C-based discrete event simulation language). Although GloMoSim provides a very desired quality for our system, the ability to simulate a network with thousands of nodes, while supporting the encounter network paradigm, it still presents the same major flaw as the tools presented before: lack of support for mobile virtual devices and mobile application testing.

QualNet [17] is an improved proprietary version of GloMoSim. QualNet is a distributed and parallel discrete event network simulation tool that can be used to simulate huge networks with heavy node interactions and it is available from Scalable Network Technologies (SNT). QualNet is built in C++, which makes the tool more accessible and easier to use.

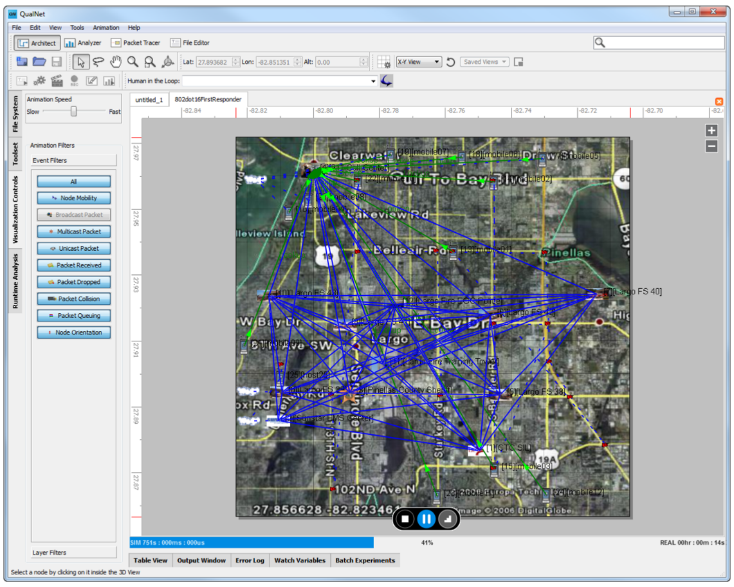


Figure 2.4: QualNet Graphical User Interface

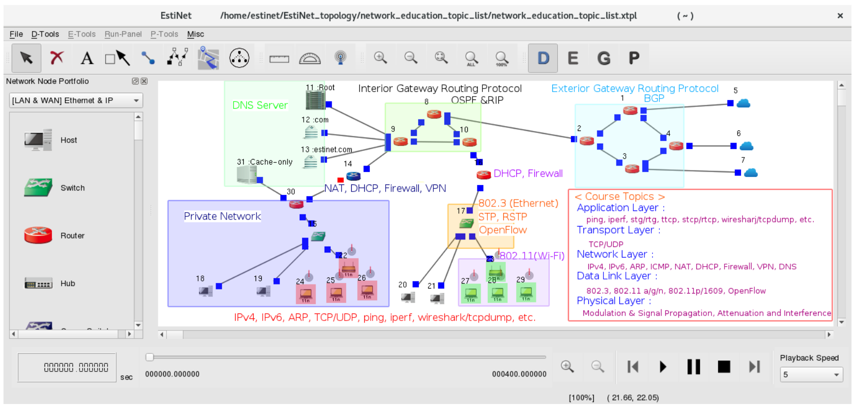
QualNet also provides a custom GUI (see Fig. 2.4) that consists of several components: i) A network analyzer that allows the user to produce statistical graphs that show network performance; ii) An architecture analyzer that shows the network design and architecture; iii) A packet tracer that serves as a visualization and debugging tool that provides a visual representation of packets being generated and exchanged throughout the network; iv) The file editor component acts as a text editing tool for various network characteristics, including number of nodes, network typologies, number of packets sent, etc. For the goals of Termite2, a good GUI would dramatically improve system usability, but when considering QualNet and its qualities, the tool ultimately fails for the same reasons as the ones presented in GlomoSim; The lack of support for mobile application testing and virtual mobile devices. Furthermore, because QualNet is only available as a commercial solution it is not very feasible when considering the project academic purpose.

We could continue discussing other Network Simulation tools, OMNET++ [18], J-Sim [19] or OP-

NET [20]. Each of these tools and the ones presented before provide distinct advantages and disadvantages between them [21] [22]. Unfortunately, a pattern can be seen throughout all of them. Despite any desirable quality these systems may have to this project, all tools fail to provide support for mobile application testing on top of the simulated network, which is crucial when considering this project fundamental objective of helping in the development of mobile application that use encounter networks.

This characteristic is directly related with the concept of network simulation itself. Network simulation tools are not concerned with what is being tested on top of the simulated network. Instead these tools focus on simulating and testing the characteristics of the network and its nodes. This fact can be shown with the system bellow.

EstiNet [23] is a network simulation and emulation tool. EstiNet is a proprietary software solution originated from the NCTUns system [24]. It is capable of simulating both wired and wireless networks, including ad-hoc networks like encounter networks. Similarly to QualNet and GloMoSim systems, it is able to simulate large networks with node execution running in parallel or in different machines. EstiNet provides a fully integrated GUI environment by which a user can edit the desired network topology, configure protocol modules and various other network functionalities (see Fig. 2.5).


EstiNet /home/estinet/EstiNet_topology/network_education_topic_list/network_education_topic_list.xtpl (~)

< Course Topics >
Application Layer :
ping, iperf, stg/rtg, ttcp, stcp/rtcp, wiresharj/tcpdump, etc.
Transport Layer :
TCP/UDP
Network Layer :
IPv4, IPv6, ARP, ICMP, NAT, DHCP, Firewall, VPN, DNS
Data Link Layer :
802.3, 802.11 a/g/n, 802.11p/1609, OpenFlow
Physical Layer :
Modulation & Signal Propagation, Attenuation and Interference

IPv4, IPv6, ARP, TCP/UDP, ping, iperf, wireshark/tcpdump, etc.

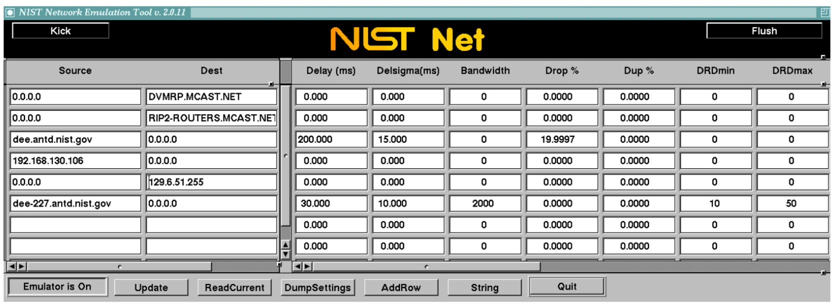
Figure 2.5: EstiNet Graphical User Interface

The major difference that this system has when compared to all others discussed before is that, because the system is also a network emulator, it provides support for application testing on top of the simulated network by allowing the network nodes to be connected with real network devices. Unfortunately, it still does not have support for physical or emulated mobile devices, thus does not provide support for mobile application testing. EstiNet is also only available as a commercial solution which makes this tool unsuitable for this work. Nevertheless, it provides insight into the types of network tools we are going to consider and present in the following section.

2.3 Network Emulation

Network emulators are normally available as hardware or software solutions that copy the behavior of a network to functionally replace it. When compared to network simulators the major difference between them is that a network emulator allows network architects, engineers, and developers to attach end-systems such as computers to the emulated network, from them to act exactly as if they were attached to a real network [25]. This allows the user to accurately gauge an application's responsiveness, throughput, and quality of end-user experience prior to applying or making changes or additions to a system. As seen in the previous section, this functionality is crucial for what we pretend to achieve with Termite2. Thus we will focus this section on discussing and presenting some popular network emulation tools, and see if they can help Termite2 satisfy its system requirements and goals.

EmuNet [26] is a network emulator fully developed in C under the Solaris operating system. EmuNet's main feature is the system ability to emulate real world packet re-routing and exchanges within the emulated network and its nodes. EmuNet achieves this by allowing each network node to capture packets and then re-route them based on a predefined re-routing table. This feature lets EmuNet recreate real life network events like network queues, traffic shaping, delay, jitter and packet loss scenarios. This allows to properly test an application and its behaviour when faced with common network problems. Unfortunately, EmuNet's main feature requires the configured network to be static, with no support for changes on network topology in run-time, which renders this system unable to properly support the encounter network paradigm where the ability to emulate node movement in run-time is crucial. Another problem derived from EmuNet main feature is the system's weak scalability, caused by the operation of real packet re-routing and exchanges. Finally the system is also unable to run virtual mobile devices and applications on top of the emulated network and lacks a proper GUI to display the emulated network. For these reasons this tool is not a suitable solution for Termite2's network emulation requirements.



Source	Dest	Delay (ms)	Delsigma(ms)	Bandwidth	Drop %	Dup %	DRDmin	DRDmax
0.0.0.0	DVMRP.MCAST.NET	0.000	0.000	0	0.0000	0.0000	0	0
0.0.0.0	RIP2-ROUTERS.MCAST.NET	0.000	0.000	0	0.0000	0.0000	0	0
dee.antd.nist.gov	0.0.0.0	200.000	15.000	0	19.9997	0.0000	0	0
192.168.130.106	0.0.0.0	0.000	0.000	0	0.0000	0.0000	0	0
0.0.0.0	129.6.51.255	0.000	0.000	0	0.0000	0.0000	0	0
dee-227.antd.nist.gov	0.0.0.0	30.000	10.000	2000	0.0000	0.0000	10	50
		0.000	0.000	0	0.0000	0.0000	0	0
		0.000	0.000	0	0.0000	0.0000	0	0

Figure 2.6: NIST Net Graphical User Interface

NIST Net [27] is a network emulator based on the Linux kernel which presents very similar abilities to the ones found on EmuNet. Similar to EmuNet, NIST Net provides the ability to perform real life network events like, network queues, traffic shaping, etc. EmuNet provides this ability by using real packet capture and re-routing while NIST Net achieves this by operating as a ”Network in a box” where the system works as a specialized router that emulates the behaviour of an entire network in a single hop, network effects such as the one presented above are done to the traffic passing through and are based on user-supplied settings. As a result of this feature the emulated network is static, not allowing any run-time modifications. This causes the system to be unable to properly emulate encounter networks. Similar to EmuNet, the system is only focused on wired networks, lacks mobile support as well as a distributed solution for improved scalability. Finally, NIST Net provides a custom GUI to monitor and change various network parameters (see Fig. 2.6). Unfortunately we can clearly see that this interface is not suitable for our project, as it does not show a visual representation of the nodes in the network and their interactions.

NetWire [28] is a network emulation system at the physical and MAC layer of the ISO/OSI network model. The ISO/OSI network model defines the model partitions of a communication system into abstraction layers, without regarding the underlying internal network structure and technology, witch makes the system extremely efficient and consistent. NetWire’s main appeal is its distributed architecture, achieved using a client/server approach to the emulated network and the applications running on top. Each client (system or application) can interact with one or more servers emulating one or more networks using the Network Oriented Emulation Language (NOEL) protocol, which is an extension of TCL over TCP/IP specifically designed for NetWire. This provides the system with a number of features. The system is able to emulate a wide range of network typologies. It has the ability to emulate basic physical characteristics of the communication channels, such as background noise. External applications can join the network (Server side) and be linked together with host adapters at any time, acting as if they were real computation nodes. Network and node emulation can be spread among multiple workstations to distribute the computational load, by connecting several network servers, which drastically improves system scalability. When comparing NetWire to EmuNet or NIST Net we can see that the system provides a clear improvement when it comes to system scalability, by using an interesting client-server approach that allows applications (clients) to join the emulated network (server) from external machines. Unfortunately when evaluating this system for our project needs, NetWire, similar to EmuNet, has no GUI, shows no support for virtual mobile devices or mobile applications and is unable to emulate encounter networks since nodes within the network are stationary. Nevertheless, on the following chapters of this work we will see that our project architecture is inspired by NetWire client-server solution, and will use this approach to separate the network emulation and the virtual mobile devices running the mobile application through multiple machines in order to improve the system scalability.

We can see that the main property of the systems presented above is the ability to perform real life network events like packet loss and delay. However, when considering network emulation for wireless networks, these events are considered normal and an expected consequence of user mobility.

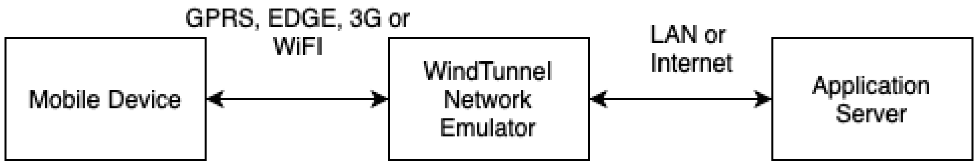


Figure 2.7: WindTunnel client-server architecture

WindTunnel [29] is a network emulator developed to perform network impact test on mobile applications in a wireless network environment. WindTunnel system is based on a client-server architecture, where mobile devices (client), and application or services (Server) can be connected, using various technologies, to Wind Tunnel emulated network (see Fig. 2.7, mobile devices can be connected via GPRS, EDGE, 3G or Wi-Fi to WindTunnel emulated network and server side application connected via the Internet or LAN.). The WindTunnel emulated network is able to simulate various network operations including, user movement and bandwidth problems. This ability to simulate user movement is one of Termite2 key requirements, unfortunately WindTunnel is designed to help users test network constrains on their applications (ex: how does the application handle a sudden drop in connection), for our project we need a network solution that takes into consideration the emulation of real world scenarios between mobile devices that translate to social interactions between users (ex: A user approaches user B, this is translated to the creation of a Wi-Fi Direct group between users A and B). Finally, WindTunnel has no support for emulated mobile devices, which goes against Termite2 requirements.

EMPOWER [30] is a scalable distributed network emulation system able to emulate wired and wireless networks with proper mobility emulation of mobile nodes in a wired network. EMPOWER main feature and advantage, if compared with WindTunnel, is the system ability to emulate multiple real network nodes in a single emulated node, granting the system its scalable quality. EMPOWER also supports the ability to emulate multi-hop operations within the emulated network, making the system fully prepared to test applications that apply MANETs as its network paradigm, where multi-hop operations are a key feature. Unsurprisingly, when considering Empower for this project objectives, the system still presents the same problems as the ones found in WindTunnel: no support for virtual mobile devices and consequently no support for automated testing of mobile applications.

EMWIN [31] is scalable IP-based mobile wireless network emulation system based on EMPOWER system. EMWIN provides all the features present on EMPOWER network emulator. The distinguishing factor between the two systems comes from EMWIN IP-based network. This lets EMWIN create conditions and traffic dynamics that emulate ”real” world packet behavior throughout the network. Packets physically traverses the network protocol stack of the emulator node's kernel, instead of logical processes in network simulators. Nevertheless, EMWIN also shows the same problems as the ones shown in EMPOWER, that make this tool unable to fulfil Termite2 desired requirements. The tool only supports physical devices connected to the emulated network and has no support for automated testing of mobile applications.

QOMB [32] is a test-bed solution created by integrating the wireless network emulator QOMET [33] with the large-scale network experiment environment StarBED [34]. The software solution provided by QOMB can be seen as the combine functionalities of QOMET and StarBED. QOMET is responsible for providing the emulated network environment and its features, which includes the ability to emulate node mobility, while StarBED is a large-scale network environment, with over 1000 standard PCs at the National Institute of Information and Communications Technology in Japan and provides the devices that populate the emulated network and are represented as network nodes. At a first glance this system may seem a possible solution for Termite2 requirements. It provides proper emulation of encounter networks and shows support for a large number of devices with a distributed architecture running on common PCs. Unfortunately this distributed architecture presents three major problems. There is no support to run automated testing tools for mobile application on QOMB platform and because QOMB can only be used on the StarBED environment, it can make accessibility difficult and limit developers options.

We can clearly see that network emulators do not fully satisfy Termite2's main requirements. Although some of the discussed systems present interesting solutions to system scalability and can properly emulate the encounter network paradigm, showing proper emulated node mobility, none of the systems show support for emulated mobile devices with the ability to run automated testing for the application being tested and developed. Another important flaw common to all discussed system is the lack of a proper GUI to help in the visualization and modeling of the emulated network, which is another of Termite2's main requirements.

2.4 Test Frameworks

In the previous section we have focused on presenting tools that target the network layer as a basis to provide support for the development of mobile application that use Encounter Networks. In this section we will now present tools that can help in the development and testing of mobile application and see if they can properly support the necessary network layer for proper development of encounter based applications.

MonkeyRunner [35] is a mobile framework primarily used in the development and creation of automated tests for Android applications. MonkeyRunner provides an API for writing programs that control an Android device or emulator from outside the Android code. Developers can use this tool to create Python programs that install android applications and then run unit test suites automatically. This includes automatic network tests but only if real devices are used, unfortunately this goes against Termite2's requirements. MonkeyRunner is not capable of emulating or simulating a network in which tests could be run on top off, but it could be integrated in a network emulation tool, in turn making this framework capable of supporting the testing and helping the development of applications that use encounter network. However, the complexity and work needed to integrate MonkeyRunner and the proper network emulation tool, would require a huge engineering effort.

There exist a lot of commercial and open source solutions to provide automated testing and helping the development of mobile application. We have tools like Appium [36], Expresso [37] and Robotium [38]. All of these tools present distinct advantages and disadvantages between them [39], and we could discuss each one of them in more detail. It is, however, unnecessary because all these tools show the same problem as the one found in MonkeyRunner. None is capable of emulating or simulating a network, which in turn renders them unable to properly execute tests for the particular case of applications that use encounter networks.

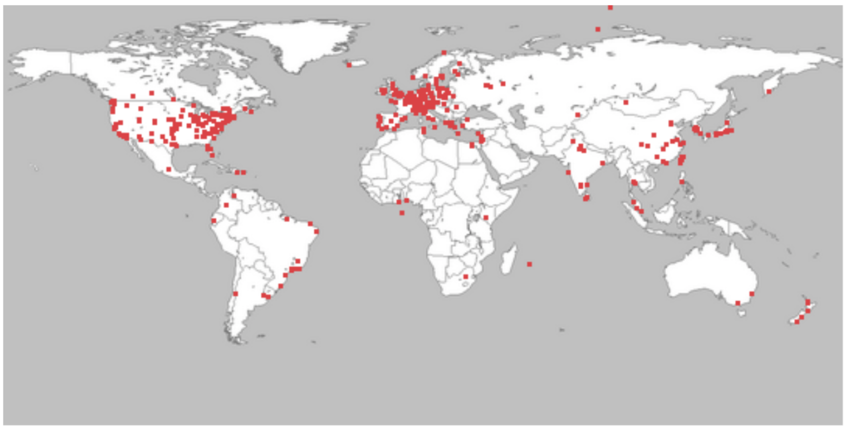


Figure 2.8: PlanetLab physical cluster of network nodes.

An interesting case can be seen with the frameworks PlanetLab [40] and EmuLab [41]. Instead of providing a tool that does automated test operations, both systems provide a network test-bed where developers can test their applications. Both system achieved this through a large physical cluster of network nodes (see Fig. 2.8, taken from PlanetLab website [40], shows the system current cluster (2020), consisting of 1353 nodes at 717 sites.), that developers can then use to execute automated application tests, including tests that focus on network operations. Unfortunately, both systems are incapable of providing a network where applications that use encounter networks could test their unique network functionalities. This happens given the physical and static nature of the clusters, which renders both systems incapable of performing crucial node activities, like node movement, crucial to the proper implementation of encounter networks.

2.5 Termite

Termite [5] [8] is an emulation test-bed for encounter networks. Termite provides support for the development and testing of mobile applications that apply the encounter networks paradigm and is based on the lack of solutions for testing and debugging of applications that target co-located moving personal mobile devices. When considering all the systems discussed before, Termite still presents itself as an obvious basis for our work. Termite is the only system that has the ability to properly test mobile application running on emulated mobile devices on top of an emulated encounter network, with proper support for node displacement and interactions. Thus, Termite is still the system that most closely resembles the one described in this document and the goals we have set. Termite shows good flexibility by presenting itself as a system capable of running on top of the most common platforms, Windows, Linux and Mac, with support for emulated Android devices. The system relies on Wi-Fi Direct to create encounter networks, which contributes to system portability, and the operation executed within the network are reproducible. Finally, the system is also robust, properly handling expected and unexpected system and user interactions.

Nevertheless, the system is not a perfect solution for our project and upon usage clearly shows some system quality problems that go against Termite2 main requirements, specially in terms of usability and system scalability.

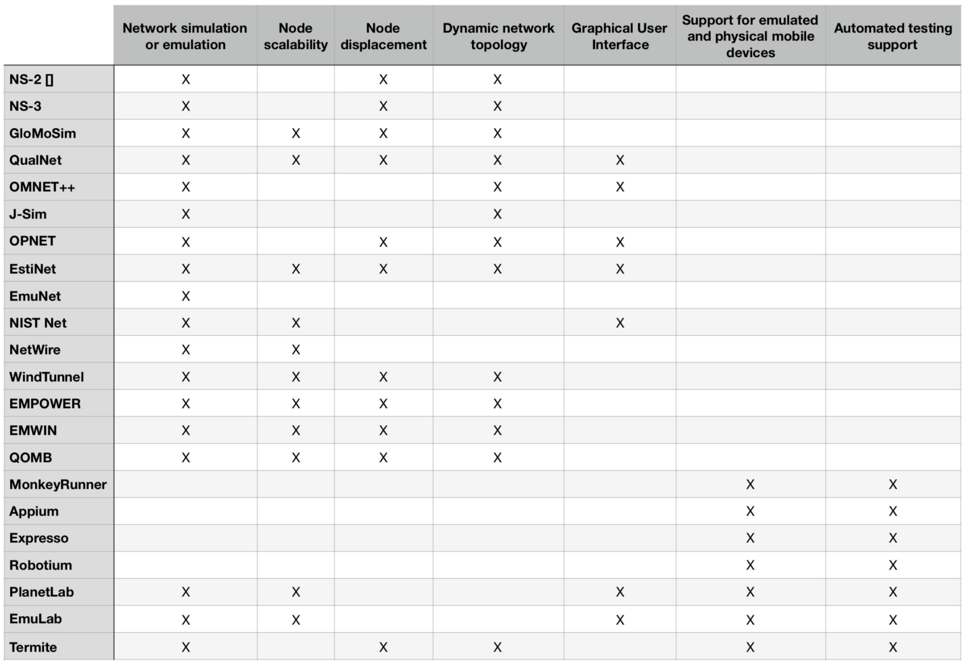
In terms of system scalability we think that Termite does not provide satisfactory results in this aspect. This happens due to three main reasons: First, is Termite's reliance on Android studio or user made scripts to create and manage all emulator instances; Second, Android SDK [42] (per default configuration) can only manage a maximum of 16 emulator instances running at the same time; Finally, emulating a large encounter network requires a large number of emulators, which requires a considerable amount of processing power to run (CPU and RAM). With this in mind, if one tries to use Termite to test an application on a large emulated network where a high number of virtual devices is required, the computer running Termite, Android Studio and all the necessary Android emulators, if not powerful enough, will show a significant slowdown. This slowdown increases with the number of devices being emulated and can even make launching new emulators impossible if the machine is severely constrained in terms of RAM. And although one could use a more powerful machine (more CPU and memory) to run more emulator instances, the user would always be limited to the maximum of 16 emulators that the Android SDK allows. Therefore, we think that the current version of Termite is only suitable for testing mobile applications on small networks and this can be seen as a major system flaw when we consider the nature of the applications that Termite is helping to develop. Termite should provide the ability to execute tests that properly emulate real world scenarios. This includes testing mobile application on small and very large networks with varying sizes of network nodes and devices interactions. Finally, it is important to note that Termite documentation says that the system is prepared to run the emulated

Android devices locally or on different machines. This feature, however, is not fully implemented, thus Termite can not present it as a solution to its scalability problems.

In terms of system usability we also think that Termite does not fully satisfy this system property. Although, Termite provides a fast and easy way to assemble the emulated network and to build mobile application tests without the need to make changes in the application code, we think that Termite presents two major flaws that severely affect the system usability. The system does not provide a proper GUI and does not allow the user to directly manage the virtual Android devices without the usage of Android studio or user made script. Termite user interface consists of the input of specific written Termite commands in a terminal window in order to produce various system operations like, node creation, node displacement, binding of emulators to virtual nodes, etc. For us this lack of a proper graphical user interface to help visualize the modeling and execution of the emulated network is a major system flaw. This is specially true when we consider the nature of the applications being tested, where the ability to see the virtual nodes moving and interacting with each other within the network is a crucial part to properly analyze the application behaviour, identify possible problems and understand the network being modeled. Finally, Termite does not offer a direct away to create and manage the emulator instances (using the terminal or a visual interface). The user is instead required to use Android Studio or their own scripts to create and manage the emulated devices. This has a sever negative impact on usability, specially when we considered a large emulated network with a large number of emulators.

2.6 Discussion

Throughout this section we presented and discussed various tools against our project goals and requirements. We have seen that most simulation Tools, although capable of supporting encounter networks, due to their nature, lack support for virtual mobile devices and the ability to test mobile applications on top of the simulated network. Network emulation tools, however, are designed with devices and application support in mind. Unfortunately, some of the tools can not properly emulate encounter networks and none of them covers the particular case of mobile applications and the necessity to support emulated mobile devices. For these reasons, all of the emulation network tools are also incapable of providing the desired solution. For the test frameworks section the opposite problem can be found. Although all the discussed tools provide the ability to execute automated tests on mobile applications with support for emulated and physical mobile devices, none of them is capable of providing the necessary network layer where tests might be done. It is important to note that for PlanetLab and Emulab frameworks, this problem does not happen. These tools main focus is to provide a real network test-bed where mobile applications can test their features. Nonetheless, these tools are also not a viable solution due to their network static nature, which makes them incapable of supporting encounter networks applications.



	Network simulation or emulation	Node scalability	Node displacement	Dynamic network topology	Graphical User Interface	Support for emulated and physical mobile devices	Automated testing support
NS-2 ▯	X		X	X			
NS-3	X		X	X			
GloMoSim	X	X	X	X			
QualNet	X	X	X	X	X		
OMNET++	X			X	X		
J-Sim	X			X			
OPNET	X		X	X	X		
EstiNet	X	X	X	X	X		
EmuNet	X						
NIST Net	X	X			X		
NetWire	X	X					
WindTunnel	X	X	X	X			
EMPOWER	X	X	X	X			
EMWIN	X	X	X	X			
QOMB	X	X	X	X			
MonkeyRunner						X	X
Appium						X	X
Expresso						X	X
Robotium						X	X
PlanetLab	X	X			X	X	X
EmuLab	X	X			X	X	X
Termite	X		X	X		X	X

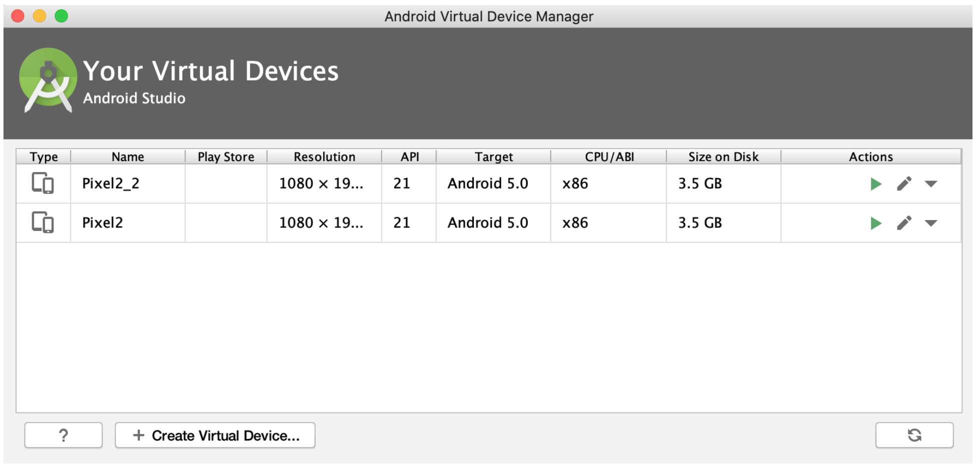
Figure 2.9: Comparison of all Related Work

Finally we discussed Termite. Although this system was planned to serve as basis for this project, it was important to also present it as related work and measure it against all other available tools. Nevertheless, five years after Termite's development, this system still presents itself as the only capable system of supporting the proper development of encounter based network applications and the only viable system to serve as basis for our project. Yet, Termite can only serve as a basis, due to the system not meeting all Termite2's requirements.

Figure 2.9 illustrates a summary of all the discussed tools and systems and how they cover Termite2's requirements. We can see that Termite is the only system that properly supports Termite2's main goals and requirements. Termite provides proper emulation of encounter networks (achieved by providing emulated dynamic network with support for node displacement) and the ability to execute automated tests of mobile application that run on emulated devices.

3 Architecture

Contents


Android Virtual Device Manager

Your Virtual Devices
Android Studio

Type	Name	Play Store	Resolution	API	Target	CPU/ABI	Size on Disk	Actions
	Pixel2_2		1080 × 19...	21	Android 5.0	x86	3.5 GB	
	Pixel2		1080 × 19...	21	Android 5.0	x86	3.5 GB	

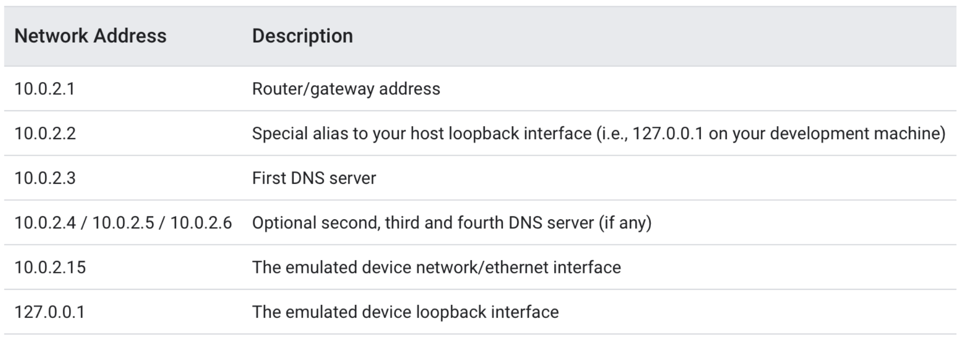
Figure 3.1: Android Studio AVD Manager.

This Chapter is divided in four main sections. We start (Section 3.1) by describing what an Android Virtual Device (AVD) is and its network environment. This is done so that we can better understand the challenges that we face, and the corresponding solutions, when we want to communicate with a mobile application running inside an Android emulator from outside the emulated environment (and vice versa). In Section 3.2 we present Termite's architecture and discuss its main components and their interactions (so that later on we can see how Termite2 differs). Then, in Section 3.3 we present our project architectural solution called Termite2; it presents a new distributed system architecture where the emulated Android devices used on the emulated network can run across multiple remote machines. Termite2 also offers the option to use a new GUI with which a user can easily create and model a network and perform the corresponding application tests. Finally, in Section 3.4 we summarise Termite2 architectural design and discuss its features when compared with the old system Termite.

3.1 Android Virtual Devices

From the Android Studio official documentation [43], an AVD is "(...) a configuration that defines the characteristics of an Android phone, tablet, Wear OS, Android TV, or Automotive OS device that you want to simulate in the Android Emulator (...)". For our project we will only consider the AVD configuration that expresses an Android phone. An AVD contains the hardware profile, system image, storage area, skin, and other properties that represent the Android device we want to emulate. With this we are then able to create multiple unique Android emulators (for our project we refer to them just as emulators) that represent the created AVD configuration.

To create and manage an emulator instance, there are two options. The first one, is to use the Android Studio AVD Manager [43]. The second, is to use the Android SDK platform commands [44]. All



Network Address	Description
10.0.2.1	Router/gateway address
10.0.2.2	Special alias to your host loopback interface (i.e., 127.0.0.1 on your development machine)
10.0.2.3	First DNS server
10.0.2.4 / 10.0.2.5 / 10.0.2.6	Optional second, third and fourth DNS server (if any)
10.0.2.15	The emulated device network/ethernet interface
127.0.0.1	The emulated device loopback interface

Figure 3.2: Android emulator virtual router address space.

options create an emulator instance on the same machine where the AVD Manager or the commands are executed.

The AVD Manager is accessed from within Android Studio, and provides the user with a graphical interface to create and manage multiple emulators and deploy mobile applications on to them. Fig. 3.1 illustrates a case in which two emulators were created, Pixel2 and Pixel2_2, both running Android 5.0 with API 21. This option to manage the emulators is much more intuitive that the command line tools provided by the Android SDK platform, but requires the user to run Android Studio. This is not ideal if the user is working on a machine with few resources or wishes to set up a large number of emulators. In fact, it is well known by all users who have ever used Android Studio, that this Integrated Development Environment (IDE) is very resource consuming.

The command line tools provided by the Android SDK platform are a set of written commands that invoke a number of tools via the terminal to perform various operations. These operations range from creating and managing the emulators, deploying and installing mobile applications on such instances and perform tests or configurations. All these tools and their command syntax is presented and explained in great detail at the official Android SDK command line documentation [44]. For our project we only need to consider the following commands: `avdmanager`, `sdkmanager` and `Android Debug Bridge (adb)`.

The `avdmanager` command allows the user to view, create and delete emulators and change the location where they are stored on the local machine. The `sdkmanager` command allows users to view, install, update, and uninstall packages for various Android SDK versions stored on their local machine. Finally, with the `adb` command, users communicate with the emulators; it facilitates a variety of actions on the emulators, such as installing and debugging apps, and provides access to a Unix shell where we can run a variety of commands and configurations directly on the device. Emulators are accessed using adb through an adb client that runs in each device.

It is also important to analyze the network environment between an emulator instance and the developer machine (when both are running in the same computer). Each emulator instance runs behind a virtual router/firewall service that isolates it from the development machine network interfaces and from

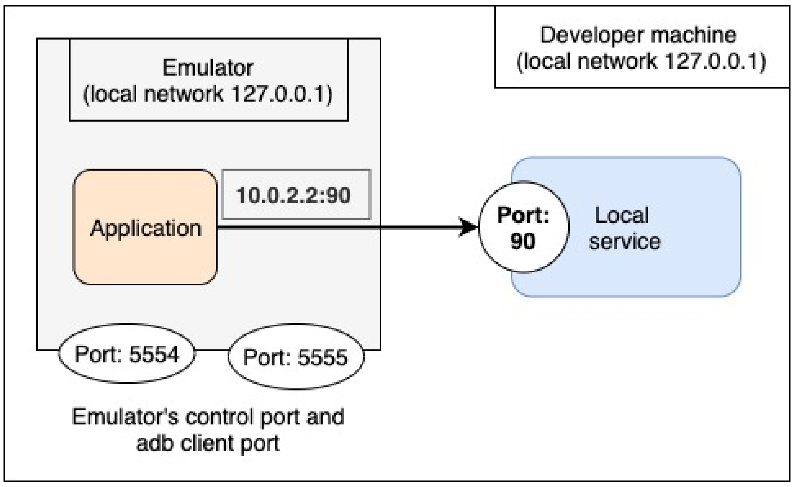

Emulator
(local network 127.0.0.1)
Developer machine
(local network 127.0.0.1)
Application
10.0.2.2:90
Port: 90
Local service
Port: 5554
Port: 5555
Emulator's control port and adb client port

Figure 3.3: Application access to some local service.

the internet. From the official Android Emulator networking documentation [45], there is a virtual router for each instance that manages the 10.0.2/24 network address space and all addresses managed by the router are in the form of 10.0.2.XX, where XX is a number.

The addresses within such address space are pre-allocated by the emulator/router and have the descriptions presented in Fig. 3.2 (this figure was taken from the Android Emulator networking documentation [45]).

To better understand how the emulator's network environment works and the impacts it has on communication, the next two figures/examples (Fig.3.3 and Fig.3.4) represent two types of communications: i) from the application in an emulator to a process running in the developer machine (3.3) ,and ii) from a client process on the developer machine to an application in the emulator (3.4).

Fig. 3.3 illustrates a scenario in which we have an application running inside an emulator instance (e.g., a smartphone app) communicating with some local service, listening in port 90 on IP 127.0.0.1 (in the developer local machine). Note that both the developer machine local network is 127.0.0.1 as well as the local network of the Emulator; this is due to the fact that emulators have their own network environment. Thus, an application running in the smartphone cannot use the address 127.0.0.1:90 to access a local service on the developer machine; in fact, this would correspond to the emulators own loopback interface (a.k.a. 127.0.0.1). Instead, the application must use the special address 10.0.2.2 (10.0.2.2:90 to access the local service).

The developer machine sees each emulator instance as a process that can be accessed via a pair of control ports (see Fig. 3.3). This pair of control ports (on the developer machine) correspond to i) a default control port, and ii) an adb client port . The default control port ranges from 5554 to 5584 (even numbers), and the adb client port ranges from 5555 to 5585 (odd numbers). Thus, each emulator instance has a control port and an adb client port pair. For example, if we start an emulator A, it will have

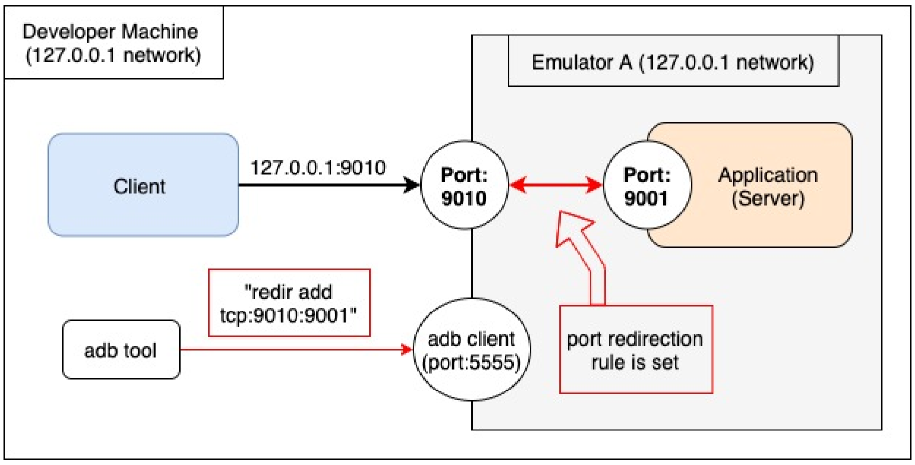

Developer Machine (127.0.0.1 network)
Emulator A (127.0.0.1 network)
Client
127.0.0.1:9010
Port: 9010
Port: 9001
Application (Server)
"redir add tcp:9010:9001"
adb tool
adb client (port:5555)
port redirection rule is set

Figure 3.4: Communication between a Client application and an application (server) inside emulator A.

the default control port 5554 and adb client port 5555; if we then start an emulator B, it will have the ports 5556 and 5557. These ports are attributed to each emulator instance until the max port numbers are reached (5584 and 5585). Due to this limited port range we can only have a maximum number of 16 emulators running simultaneously on the same developer machine.

It is important to note that the ports mentioned above only grant access to the emulator instance and not to the mobile application that might be running inside them; such ports are meant only to detect and configure the emulator instance. The communication between a process on a developer machine network environment and an application running inside an emulator instance, requires that we first set a port redirection rule on the emulator.

To better understand how such redirection can be done, imagine that we wish to connect a client application, running on the developer machine, to an application, acting as a server, inside emulator A that is listening on port 9001 (see Fig. 3.4).

As previously described, we know that processes (in this case the client application) outside the emulators network environment cannot reach or locally access the server application running inside it. To achieve this, we first need to set a port redirection rule on the emulator. This is done by using the adb command line tool (that communicates with the emulator through the adb client port) to perform a port redirection command, `redir add tcp:X:Y` that ”tells” the emulator A that any connections received on the port X in the localhost network of the developer machine must be redirected to the port Y inside the localhost network of the emulator A (X can be any available port number on the localhost network of the developer machine and Y is a port number of the application listening inside the emulators network environment). Assuming X = 9010 and Y = 9001, Fig. 3.4 shows a representation on how this port redirection is set (red arrows) and the communication that it enabled. The client accesses the server application inside the emulator A using the developer machine address 127.0.0.1:9010 which is redirected to the server port 9001 inside the emulators localhost network (black arrow).

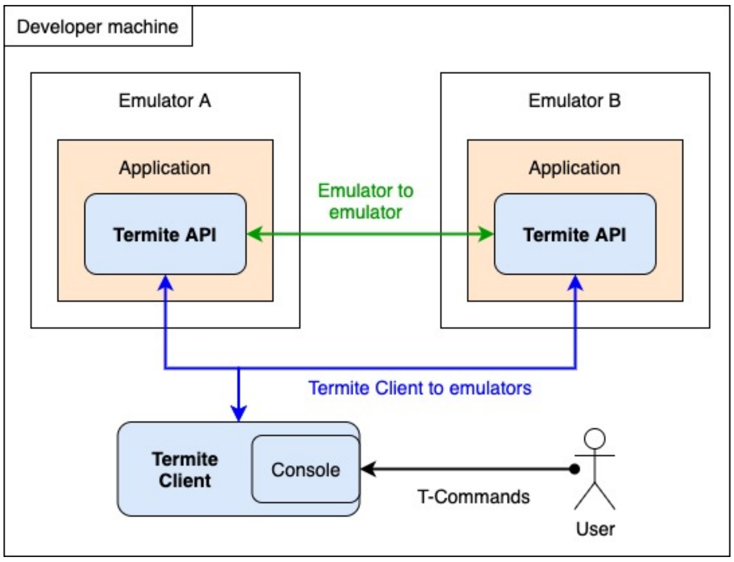


Figure 3.5: Termite's components and their interactions.

3.2 Termite Background

As mentioned before in Chapter 2.6, the current Termite system is the starting point for our work. Thus, it is important to briefly describe Termite's system architecture and its components.

Termite is a test-bed solution that helps users to develop and test their mobile Android applications that use encounter networks. To this end, Termite allows a user to emulate an encounter network where each virtual node can be bound to an emulator running the application being developed/tested (all emulators must run on the same machine as Termite and when bound to virtual nodes we called them target emulators). On this network the user is then able to perform various peer-to-peer scenarios among the target emulators.

The Termite system is comprised by the interaction of two system components, the Termite Client and Termite API. Termite Client is Termite's main component; it starts from a console window on the developer machine and it is through here that all user interactions with the system are done using Termite commands (we call them T-Commands) including the creation and modelling of the emulated network. The Termite API is an Android library (developed for Android 5.1 or above, allowing the library to be used on more than 92% of all Android devices and application) that implements most of the WiFi-Direct API and must be used by the developer on the applications that he/she wishes to develop/test using Termite. These applications run on emulators on the developer machine.

There are two types of interactions between these system components: Termite Client to emulators, and emulator to emulator communication. Note that with the current Termite, all these interactions happen locally on the developer machine (localhost). Fig. 3.5 shows a simplified depiction of these components and their interactions.

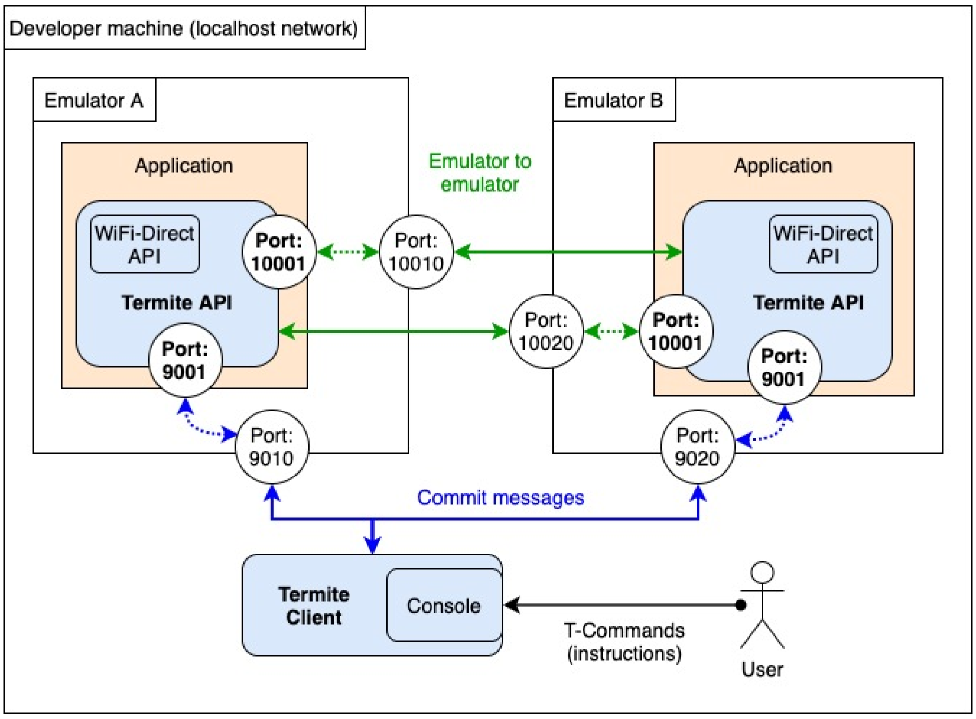


Figure 3.6: Termite API socket servers and communication channels.

In the following sections we explain Termite's components in more detail and how the above mentioned interactions do occur.

3.2.1 Termite API

As we discussed in Section 1.1, the current implementation of the WiFi-Direct API [46] on Android does not allow the use of emulators to develop and test encounter-based applications. This happens because it is not possible to emulate the peer-to-peer scenarios using the available WiFi-Direct API on emulators. Termite API solves this by generating the same WiFi-Direct peer-to-peer events on the emulators to those that would happen when using the WiFi-Direct API on real physical Android devices.

Such events correspond to the detection that other emulators are in WiFi-Direct range or that a peer-to-peer group was created with other emulator(s). For more information on these events please see the WiFi-Direct API documentation [46] or the Termite API [8].

When in use, the Termite API runs a background socket server that continually listens on port 9001 (inside the emulator network environment). Commit messages are received on this socket following instructions given by the user in the Termite Client. Commit messages are sent in two cases: when nodes (within the emulated network) move close to others, or when peer-to-peer groups are formed between nodes (also within the emulated network).

The interaction between the Termite Client and the target emulators is done by commit messages.

These messages contain information that allows the Termite API to trigger the proper peer-to-peer events inside an application using the WiFi-Direct API. For example (see Fig. 3.6) a group is created inside the emulated network between two target emulators A and B; then, a commit message is sent to each one (blue arrows).

Termite Client sends this commit message using the addresses localhost:9010 for Emulator A and localhost:9020 for emulator B. The connection to these addresses is then redirected to the Termite API socket server on port 9001 inside each emulator (this and the following redirection are discussed on the next Section 3.2.2 and are represented on Fig. 3.6 by the doted arrow lines).

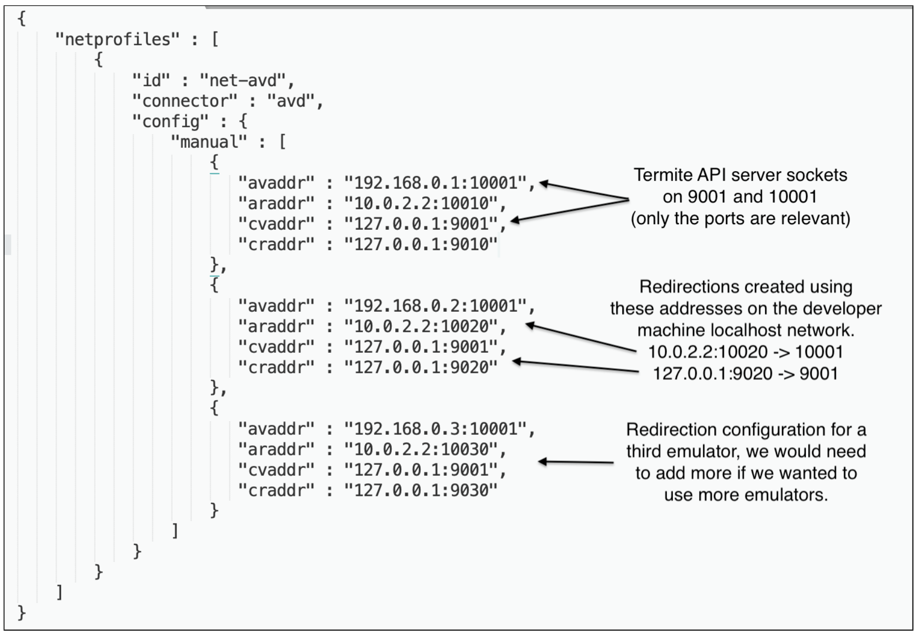
When a commit message is received by the Termite API (on port 9001, represented on Fig. 3.6 by the doted blue arrows) it triggers an Android broadcast inside the application that informs it that a peer-to-peer group was created with another emulator. When a group is created between two target emulators, the Termite API allows each target emulator to receive incoming communications from each other by using a socket server on port 10001 (running within each of the target emulators). Emulator A connects to Emulator B using the address 10.0.2.2:10020 and Emulator B connects to Emulator A using the address 10.0.2.2:10010 (10.0.2.2 refers to the localhost of the developer machine as seen in Fig. 3.2 from Section 3.1). Both of this connections are then redirected to the server socket 10001 opened by the Termite API inside each emulator's network (the addresses presented above are obtained from the commit message).

3.2.2 Termite Client

Termite Client has two main responsibilities: i) to create and model an emulated encounter network using Termite Nets (T-Nets are responsible for the layout of the emulated network and it is well documented in Termite [5]), and ii) set the necessary port redirection rules on the target emulators.

The port redirection rules are set manually by the user through the Termite Client Console (Termite Client then uses the Android SDK adb tool)and are set on each emulator by associating ports inside the emulators localhost network to ports on the developer machine localhost network. In the case of Termite, the internal ports used correspond to the socket servers opened by the Termite API and the outside ports are chosen by the Termite Client with the help of a configuration file, configured by the developer (the file is stored inside Termite Client and an example of this file is shown on Fig. 3.7). In Fig. 3.8, the red arrows and text represent the port redirection operations.

The local addresses generated by setting these rules are then used by the Termite Client to identify each target emulator and are used to establish the communications discussed before. For example, in Fig 3.8, the target emulator A is identified by the addresses localhost:9010 and localhost:10010 and the target emulators B by the addresses localhost:9020 and localhost:10020. Termite Client to target emulators communication is done through the addresses localhost:90X0 (that redirects to the port 9001 inside



```
{
    "netprofiles" : [
        {
            "id" : "net-avd",
            "connector" : "avd",
            "config" : {
                "manual" : [
                    {
                        "avaddr" : "192.168.0.1:10001",
                        "araddr" : "10.0.2.2:10010",
                        "cvaddr" : "127.0.0.1:9001",
                        "craddr" : "127.0.0.1:9010"
                    },
                    {
                        "avaddr" : "192.168.0.2:10001",
                        "araddr" : "10.0.2.2:10020",
                        "cvaddr" : "127.0.0.1:9001",
                        "craddr" : "127.0.0.1:9020"
                    },
                    {
                        "avaddr" : "192.168.0.3:10001",
                        "araddr" : "10.0.2.2:10030",
                        "cvaddr" : "127.0.0.1:9001",
                        "craddr" : "127.0.0.1:9030"
                    }
                ]
            }
        }
    ]
}
```

Termite API server sockets on 9001 and 10001 (only the ports are relevant)
Redirections created using these addresses on the developer machine localhost network. 10.0.2.2:10020 -> 10001 127.0.0.1:9020 -> 9001
Redirection configuration for a third emulator, we would need to add more if we wanted to use more emulators.

Figure 3.7: Termite Client port redirection configuration file.

the emulator) and the addresses localhost:100X0 are used for emulator to emulator communication (that redirects to the port 10010 inside the emulator).

3.3 Termite2

Our architectural solution was designed to solve Termite's scalability and usability problems. To achieve this, we developed a new version of the Termite system, called Termite2. Termite2 and Termite have the same goal: to help users to develop and test their mobile encounter applications However, Termite2 present two major system differences when compared with Termite: i) Termite2 is able to use emulators that are running across multiple machines, and ii) Termite2 has a new graphical user interface (in addition to the console) that runs on a web browser and displays a visual representation of the emulated network being created and allows the user to interact with it via clicks and menu choices.

These system differences are expressed on Termite2 architecture through the creation of two new system components: the Termite2 Server, and the Termite2 GUI. These new components work alongside the old Termite's components, Termite Client and Termite API, now called Termite2 Client and Termite2 API on Termite2. Therefore, Termite2 is comprised by the interaction of the following components: Termite2 Client, Termite2 Server, Termite2 API and Termite2 GUI (see Fig. 3.9).

Termite2 Client runs from a console window on the developer machine where the emulated network is created and modelled (following the T-Nets model). User interaction with Termite2 is also performed through this Client, however the user is now able to choose between two interface options: the old

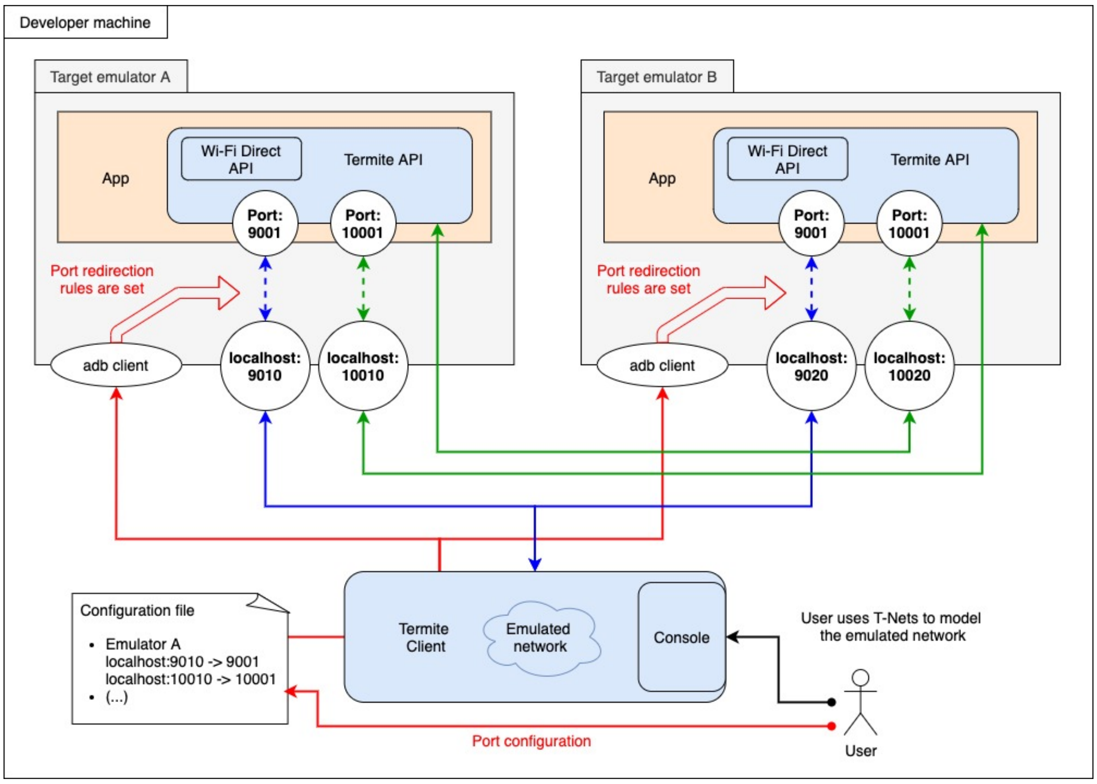


Figure 3.8: Termite Client port redirection rules and connections.

console interface where system interactions are done using T-Commands, or a new GUI where system interactions are performed via interactive menus and options.

Termite2 API is an Android library (developed for Android 5.1) with the same responsibilities and features as the ones found on Termite API (Termite system).

There still exists two types of communication to consider: from Termite2 Client to emulators, and emulator to emulator. However, these communications are no longer performed in the same way as they were on Termite. Fig. 3.9 shows the differences: Termite2 Client communicates with the emulators through the Termite2 Servers (blue arrows); the same happens for emulator to emulator communication (green arrows).

We assume that Termite2 Server(s) run in the same network as the Termite2 Client (this can be overwritten if we use software that can make remote networks appear as local networks and on Section 5 we show and use a tool called sshuttle [47] [1] which does exactly this). There must be a Termite2 Server running on each local machine in which we want to run emulators on. In order for Termite2 Client to access a Termite2 Server (and the emulators managed by it) the user must register them. This consists of writing the local IP addresses of the machines where Termite2 Servers are running on a file inside Termite2 Client (called configuration file in Fig. 3.9).

[1] sshuttle allows us to forward the IP addresses and ports of an entire remote network to a local network

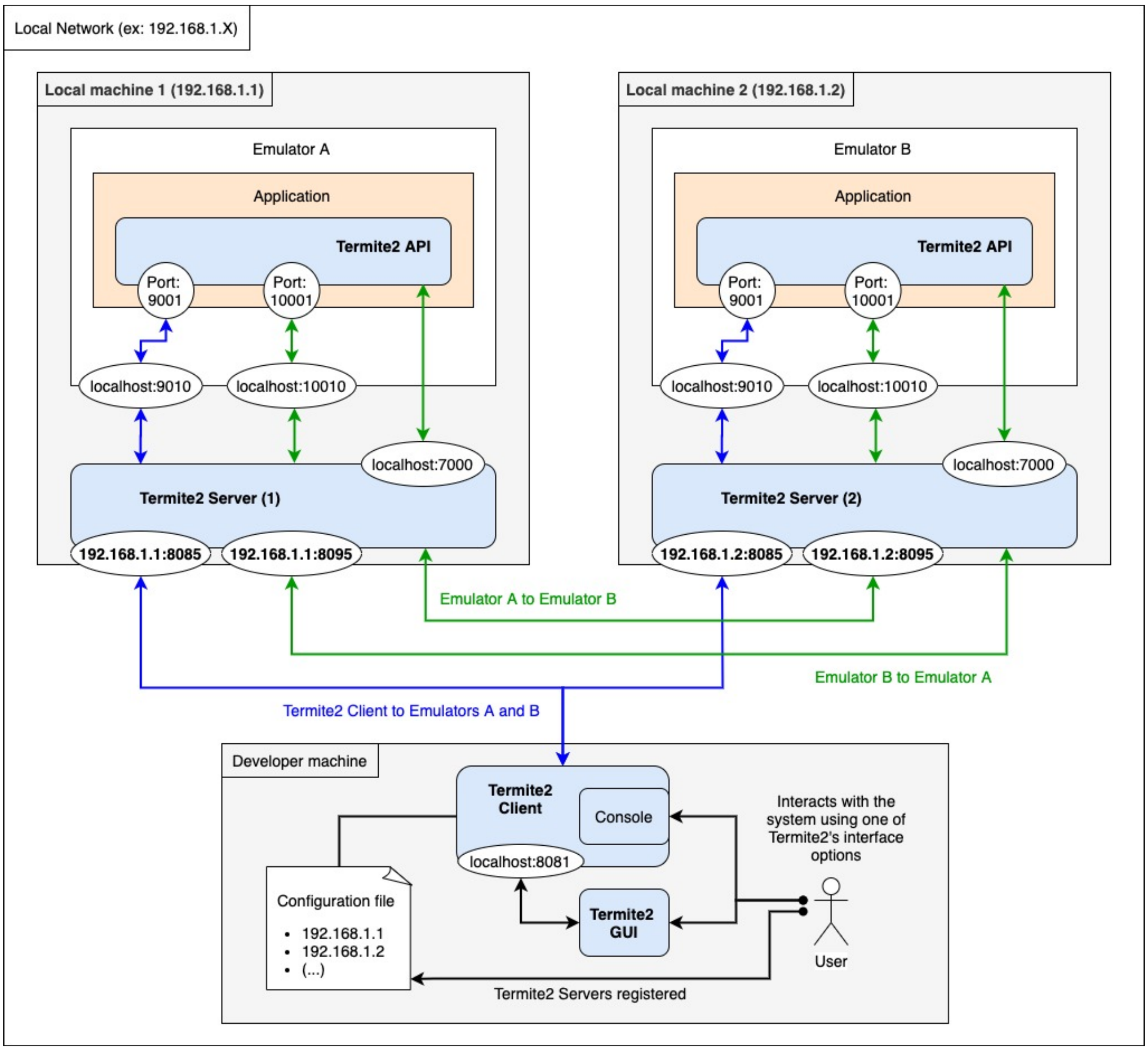


Figure 3.9: Termite2's components and their interactions.

In the following Sections we describe Termite2's new components (Termite2 Server and Termite2 GUI) in more detail, and their interactions.

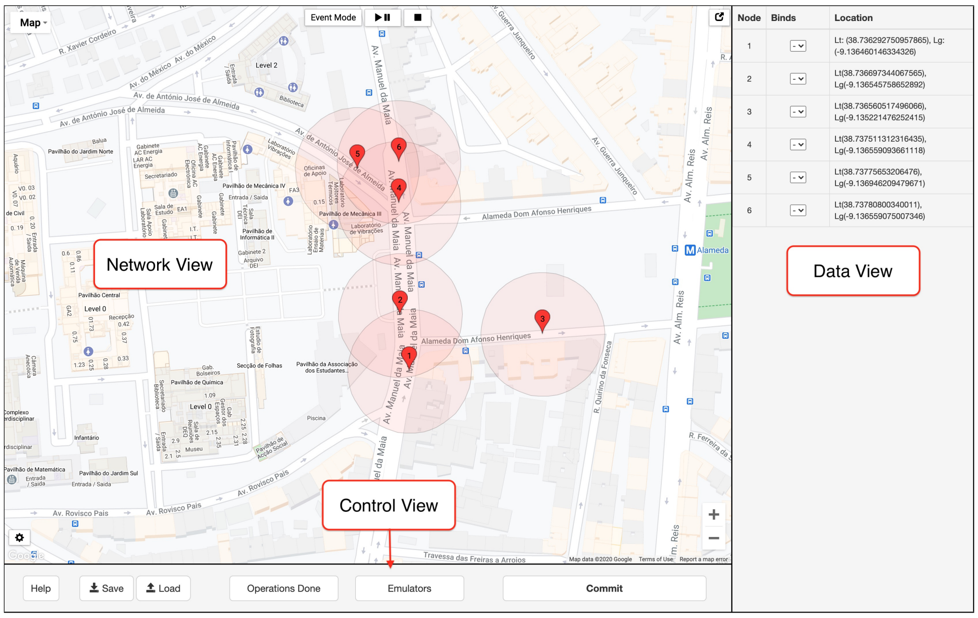
3.3.1 Termite2 Server

Thanks to the new Termite2 Server component, Termite2 is now able to support a much larger number of emulators, distributed across multiple machines. In particular, Termite2 Server identifies the emulators (running locally) the same way as Termite Client did. This is done using the addresses created when setting the redirection rules on each emulator. However, with the Termite2 Server the redirection rules are set now automatically and the user no longer needs to specify the address/ports to be used on a configuration file. Another important change is that emulators are now also identified by the machine IP address where they are running. For example in Fig. 3.9, the emulator A is identified by the addresses localhost:9010, localhost:10010 and the local machine 1 network address (192.168.1.1); emulator B is identified by the addresses localhost:9010, localhost:10010 and the local machine 2 network address (192.168.1.2).

As shown in Fig. 3.9, Termite2 Client connects with each registered Termite2 Server through the addresses 192.168.1.1:8085 and 192.168.1.2:8085. Note that the port value 8085, is predefined on the Termite2 Server but can be changed to any other port value chosen by the user by configuring a `connectionports.txt` file inside Termite2 Server source folder. With these connections, Termite2 Client is then able to discover and use the emulators that are running on each Termite2 Server machine. These connections are also used by the Termite2 Client to send the commit messages (upon user interaction) to the emulators using the Termite2 Servers as intermediaries (blue arrows in Fig. 3.9).

Like in Termite system, emulators can communicate with each other when a commit message is received (using the Termite2 API server socket on port 9001), informing the emulator/application that a peer-to-peer group was created. When this happens, the application can then use the Termite2 API to create a socket connection with the group member (the socket connection is done using the addresses of each emulator, provided within the commit message information). However, this connection is now performed with the Termite2 Server(s) of each emulator as intermediaries. In Fig. 3.9 we can see this and the addresses used by looking at the green arrows. The Termite2 API on the emulator A connects to the emulator B (on 192.168.1.2:10010) by first opening a connection to the local Termite2 server (1) using the address localhost:7000 (the port value 7000, is a fixed and unchanged value, known by the Termite2 API in order for the connection with the Termite2 Server to be possible). When this connection is made, Termite2 API registers that the emulator A wants to communicate with the emulator B on address 192.168.1.2:10010. The Termite2 Server (1) opens a connection to the Termite2 server (2) using the address 192.168.1.2:8095 (the port value 8095 is predefined but similar to the port 8085 value for the Termite2 Client connections, it can be changed by the user through a configuration file inside the Termite2 Server source folder). When the connection is made, the Termite2 Server (2) registers that a remote emulator wants to communicate with the local emulator B on port 10010. This Termite2 Server (2) then opens a connection with the emulator B using the address localhost:10010. After all these redirections the applications on both emulators are able to communicate. None of these redirections are seen by the user.

Lastly, it is important to present another capability of Termite2 Server. Termite2 Server allows the user to manage the emulators life cycle (create, destroy, start, stop and on them install and start applications) from the Termite2 Client components. This is crucial for Termite2 scalability as it allows the user to create and manage a large number of emulators (and the applications running inside) distributed across multiple machines from a single control point. The commands used to perform this task can be found on



Node	Binds	Location
1	-	Lt: (38.736292750957865), Lg: (-9.136460146334326)
2	-	Lt(38.736697344067565), Lg(-9.136545758652892)
3	-	Lt(38.736560517496066), Lg(-9.135221476252415)
4	-	Lt(38.737511312316435), Lg(-9.136559093661118)
5	-	Lt(38.73775653206476), Lg(-9.136946209479671)
6	-	Lt(38.73780800340011), Lg(-9.136559075007346)

Figure 3.10: Termite2 GUI views.

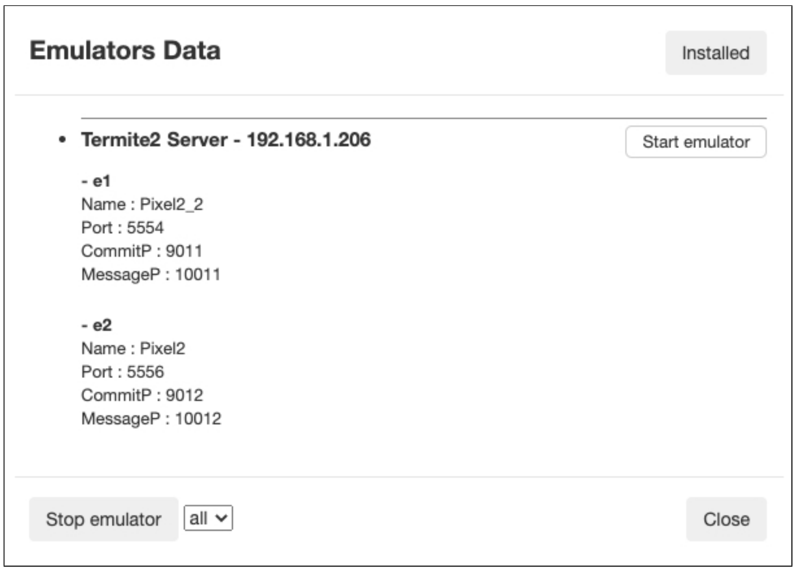
the Appendix B of this document. Later on, when addressing Termite2 evaluation (see Section 5) we show how this feature allows us to deploy a large number of emulators, spread across multiple machines, in a small amount of time, contributing to the scalability of Termite2.

3.3.2 Termite2 GUI

Termite2 GUI is a new interface option provided by the Termite2 Client. This interface runs inside a web server on the developer machine and can be accessed via a web browser. In order for the Termite2 GUI to send and receive data from the Termite2 Client it connects to it through socket server that run on the Termite2 Client on localhost:8081 (we show this connection on Fig. 3.9 in Section 3.3).

Visually, this new interface is divided in three distinct parts (see Fig. 3.10). The first one is the Network view, where the user is able to create and model the emulated network on a real world map (provided by Google Maps) through interactive mouse clicks, drags and menu choices. The virtual network nodes are also presented on this view (red markers on Fig. 3.10). The second part is the Data View, where the user is able to see relevant data associated with the virtual nodes created on the network. This data includes virtual node name, geographical location (coordinates) and what emulator is the virtual node bound to (node bind is also done here through a simple select menu); information regarding the peer-to-peer groups created on the emulated network can also be seen on this view (group owner and group members). Finally, the Control View is comprised by a set of buttons that allow the user to perform a number of tasks on the interface (most of them self explanatory based on the buttons' names).

The most interesting task that one should be aware of on the Control View is how emulators can be



- **Termite2 Server - 192.168.1.206**

- e1
Name : Pixel2_2
Port : 5554
CommitP : 9011
MessageP : 10011

- e2
Name : Pixel2
Port : 5556
CommitP : 9012
MessageP : 10012

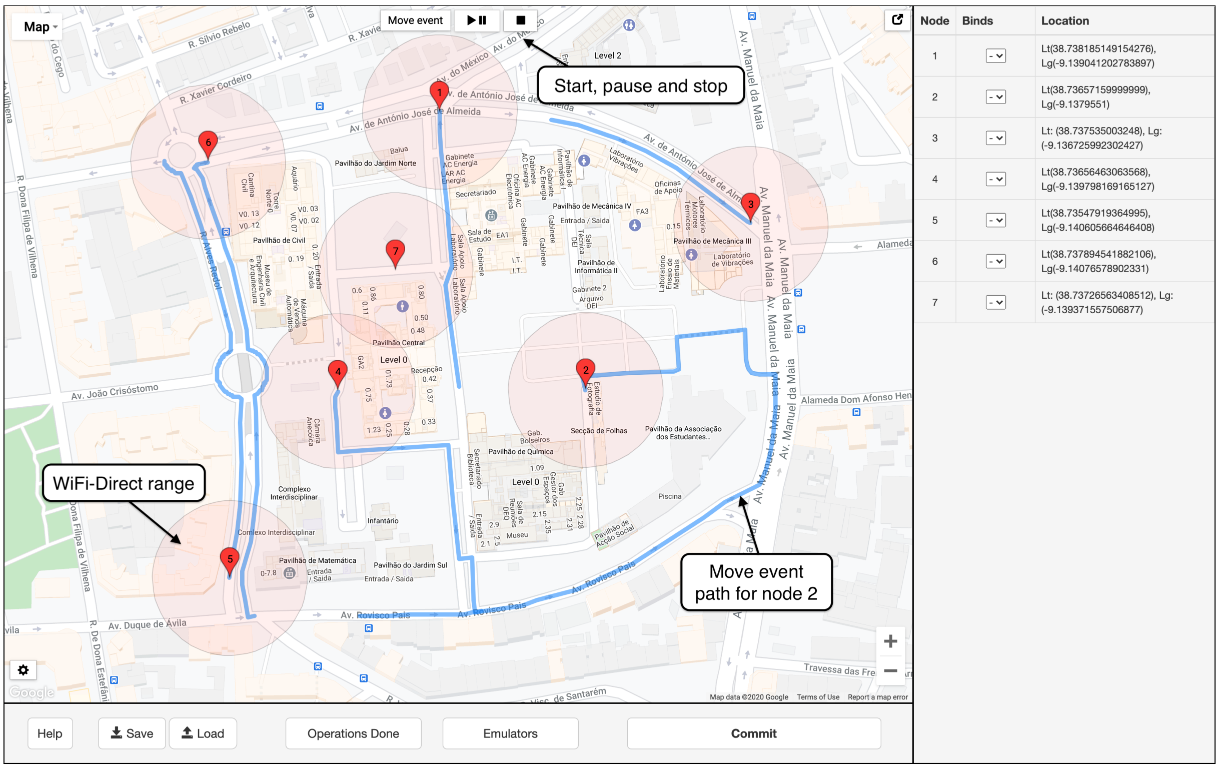
Figure 3.11: Emulators sub-view on the Termite2 GUI.

managed through a sub-view accessed by clicking the Emulators button (see Fig. 3.11). On this sub-view, the user is able to see all connected Termite Servers and the emulators that are managed by each one (in the Fig. 3.11 we see that we are connected to the Termite2 Server on the network 192.168.1.208 where 2 emulators, e1 and e2, are running). We are also able to start multiple emulators by pressing the button ”Start emulator”, choosing how many emulators we want to start and what application we want to automatically start on each. Lastly, at the bottom left we have the button ”Stop emulator” that lets us stop any of the emulators shown.

Also note that we can emulate node movement on the emulated network when we use the new interface. On Termite2 GUI we are able to emulate node movement through two distinct actions. We can simply drag and drop a node to a new location or move nodes through an automatic movement event that we call move/movement events. Movement via drag and drop is the simplest way to emulate node movement; however, when creating complex tests (with a large number of nodes moving) this movement method is not ideal (we can only drag one node at a time) and the user should instead use move events.

Move events allow the users to create automatic movement paths for each network node with a fixed movement speed that emulates walking, cycling or driving (see Fig. 3.12). When nodes move via these events they automatically create/join peer-to-peer groups when in range of other nodes (node range is displayed by the red circular area around each node and emulates WiFi-Direct range). Move events are started, paused or stopped through the buttons at the top of the Network View and allow the user to create more natural and complex movement scenarios that better emulate real world interactions.

These events are performed on the interface and internally (on the interface logic) generate a sequence of commands similar to those that one would have to use to model the same scenarios on the



Node	Binds	Location
1	-	Lt(38.738185149154276), Lg(-9.139041202783897)
2	-	Lt(38.73657159999999), Lg(-9.1379551)
3	-	Lt: (38.737535003248), Lg: (-9.136725992302427)
4	-	Lt(38.73656463063568), Lg(-9.139798169165127)
5	-	Lt(38.73547919364995), Lg(-9.140605664646408)
6	-	Lt(38.737894541882106), Lg(-9.14076578902331)
7	-	Lt: (38.73726563408512), Lg: (-9.139371557506877)

Figure 3.12: Termite2 GUI move events.

console. When an event is stop/finished the user can then perform a commit operation (using the commit button that works similarly as a commit command on the console) that propagates the full event to the target emulators in order to test the application on the created scenario. When a commit is performed, a commit message is set via a socket connection to Termite2 Client on localhost:8081. Message are received by the Termite2 Client and sent to the target emulators.

3.4 Summary

This section shows the architecture of Termite2. When compared to the previous (and current) version, called Termite, Termite2 architecture supports improved scalability and usability.

Put simply, Termite2 improves scalability by adding a new component called the Termite2 Server that changes how the Termite2 Client interacts with the emulators. Termite2 Server(s) allows the Termite2 Client to use multiple emulators distributed across multiple machines on the emulated network. This not only allows users to create large emulated networks where complex tests can be created (we are no longer limited to a maximum number of 16 emulators, like we were on Termite) but also supports a quicker development and testing of applications (when compared to the current version of Termite).

Emulator identification was also improved as it is now done automatically by the Termite2 Server and no longer requires the user to configure the ports to be used on the redirection rules. Managing the emulators life cycle (emulator creation, deletion, start, stop and on them install and start applications) can now be done from Termite2's system itself from a single control point (the Termite2 Client) without the need to use Android Studio or user made scripts.

Finally, Termite2 improves Termite's system usability by providing a new interface option, the Termite2 GUI. This new graphical interface allows a user to see the emulated network being created and all the node interactions that happen within on a real world map. Modeling the emulated network is now done by using interactive clicks and menu choices (this includes the deployment and creation of the target emulators). Termite2's new GUI also support new movement options that allow the user to more easily emulate node movement and create complex testing scenarios.

4
Implementation

Contents

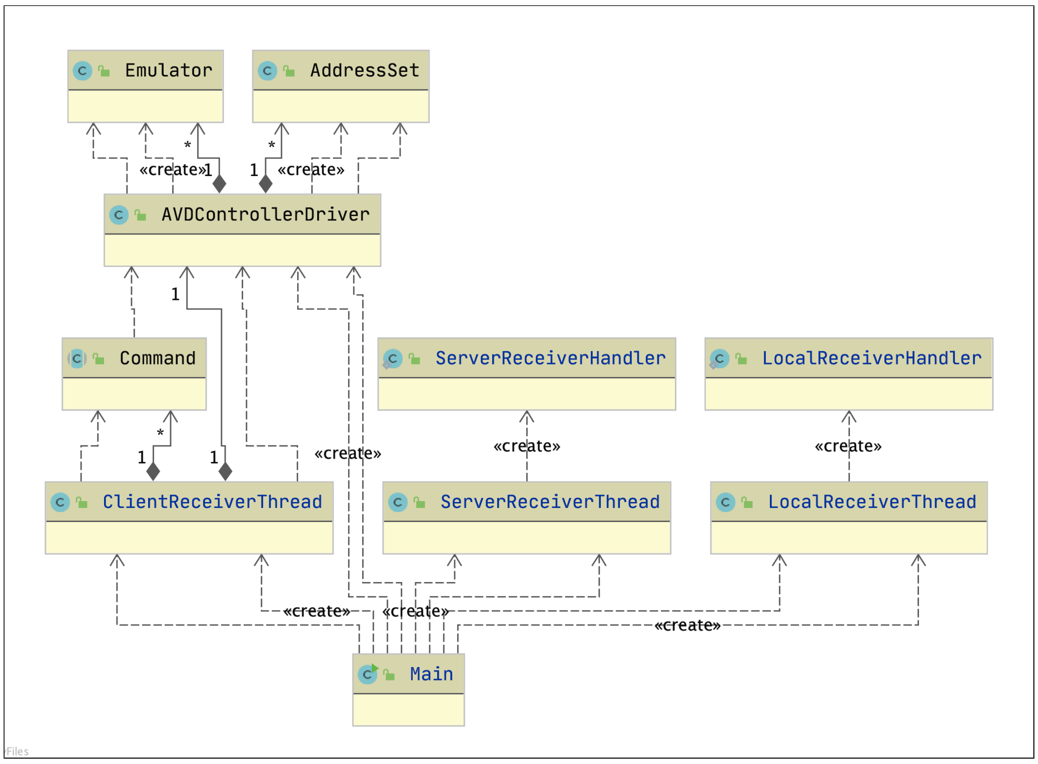


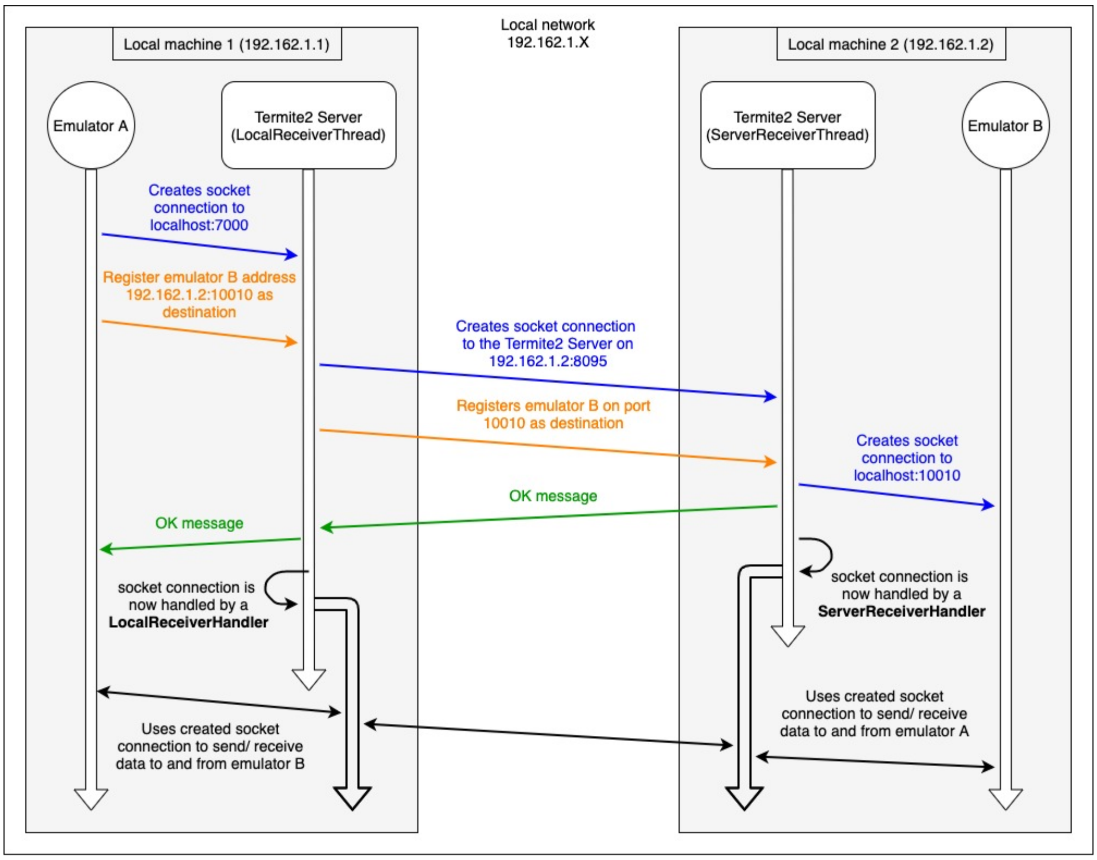
Figure 4.1: Simplified UML diagram of Termite2 Server Java classes.

In this chapter, we discuss the implementation of Termite2 components: Termite2 Server, and Termite2 GUI. We do not dive deep into the code implementation of each component (as this would be far too long and unnecessary); instead, we focus on the technologies used and how the code is structured in other to implement Termite2.

4.1 Termite2 Server

The implementation of the Termite2 Server was done using Java. We use this language to guarantee that this component can easily run on Windows, Mac or Linux machines and to be consistent with the rest of Termite code base (which is already in Java). Fig. 4.1 shows a Unified Modeling Language (UML) diagram of the Termite2 Server Java classes (for simplicity we only show the most relevant classes and no fields are presented).

The AVDControllerDriver class contains all the necessary methods that must be used to interact and configure the emulators. Its methods leverage the available Android SDK tools and allow for the creation, deletion, start and stop of new emulators and the installation/start of applications on them. This class also provides the necessary methods to detect the online emulators and set the necessary port redirection rules. The addresses used to set the redirection rules are created and managed by the



Local network
192.162.1.X
Local machine 1 (192.162.1.1)
Local machine 2 (192.162.1.2)
Emulator A
Termite2 Server (LocalReceiverThread)
Termite2 Server (ServerReceiverThread)
Emulator B
Creates socket connection to localhost:7000
Register emulator B address 192.162.1.2:10010 as destination
Creates socket connection to the Termite2 Server on 192.162.1.2:8095
Registers emulator B on port 10010 as destination
Creates socket connection to localhost:10010
OK message
OK message
socket connection is now handled by a LocalReceiverHandler
socket connection is now handled by a ServerReceiverHandler
Uses created socket connection to send/ receive data to and from emulator B
Uses created socket connection to send/ receive data to and from emulator A

Figure 4.2: LocalReceiverThread and ServerReceiverThread operation.

AddressSet class. When the redirection rules are set on an emulator, an Emulator object is created that identifies it.

The ClientReceiverThread class is responsible for continually listening for Termite2 Client connections on a server socket on port 8085 (on the machine local network where Termite2 Server is running, as show on the previous Section 3.3.1). When a connection is received, the ClientReceiverThread registers the connection and is now ready to receive the Termite2 Client requests and redirect possible commit messages to the target emulators. Termite2 Client requests are processed by Command classes that express the operations received, for example: sending information about what emulators are available to be used on the emulated network, redirecting commit messages to the emulators, or starting a new emulator instance.

Finally, we have the LocalReceiverThread and the ServerReceiverThread classes. These classes are responsible for handling the redirections that happen when an emulator communicates with another (see, Fig. 4.2). As we presented on Section 3.3.1 an emulator A communicates with an emulator B (on the network 192.168.1.2) by first opening a socket connection to the local Termite2 Server on localhost:7000. The Termite2 Server then opens a socket connection to the Termite2 Server of emulator B (this is implemented on by the LocalReceiverThread class). The Termite2 Server of emulator B receives this connection and then opens a socket connection to emulator B (this is implemented on the Server-ReceiverThread). When all these socket connections are established, both the LocalReceiverThread and ServerReceiverThread classes create handler threads (LocalReceiverThreadHandler and Server-ReceiverThreadHandler) that handle any subsequent messages that might be exchanged between emulators A and B. This is done in order to guarantee that Termite2 Server can handle the redirection of multiple communications between emulators.

4.2 Termite2 GUI

The Termite2 GUI is built to run inside Apache Tomcat Server. The interface logic is done using JavaScript and the emulated network is displayed on a real world map using the Google Maps API [48]. We choose JavaScript as it allows the interface logic to run across any modern web browser (Google Chrome, Firefox, Safari, etc.) and Google Maps API due to its extensive documentation and features.

As with Termite2 Server, if we consider the whole implementation of Termite2's GUI, it is not feasible to present all the code, and it would be too long and unnecessary as well. Most of the code used translates the already existing command logic on the Termite console to interactive button clicks and menu choices. Thus, we consider that the only implementation aspects of the GUI that makes sense to present in this section are those related to: i) the way information is exchanged between the Termite2 GUI and the Termite2 Client, and ii) the new automatic node movement option (presented in the previous Section 3.3.2).

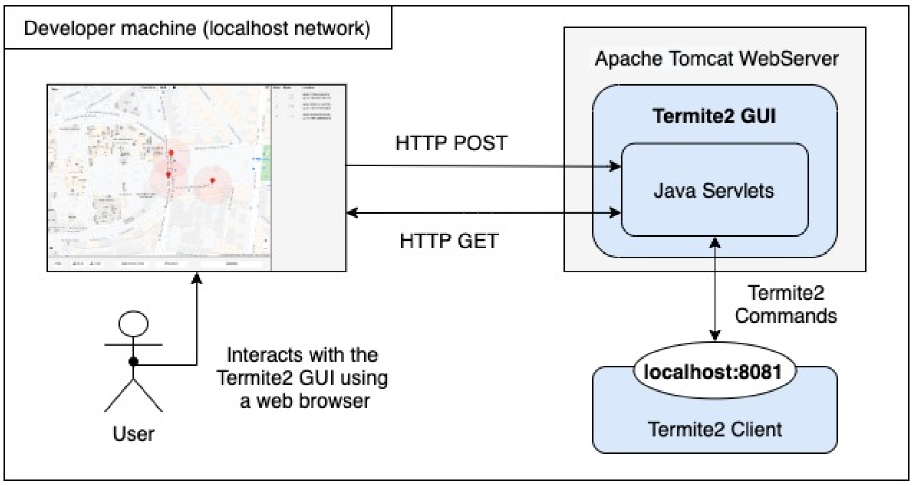
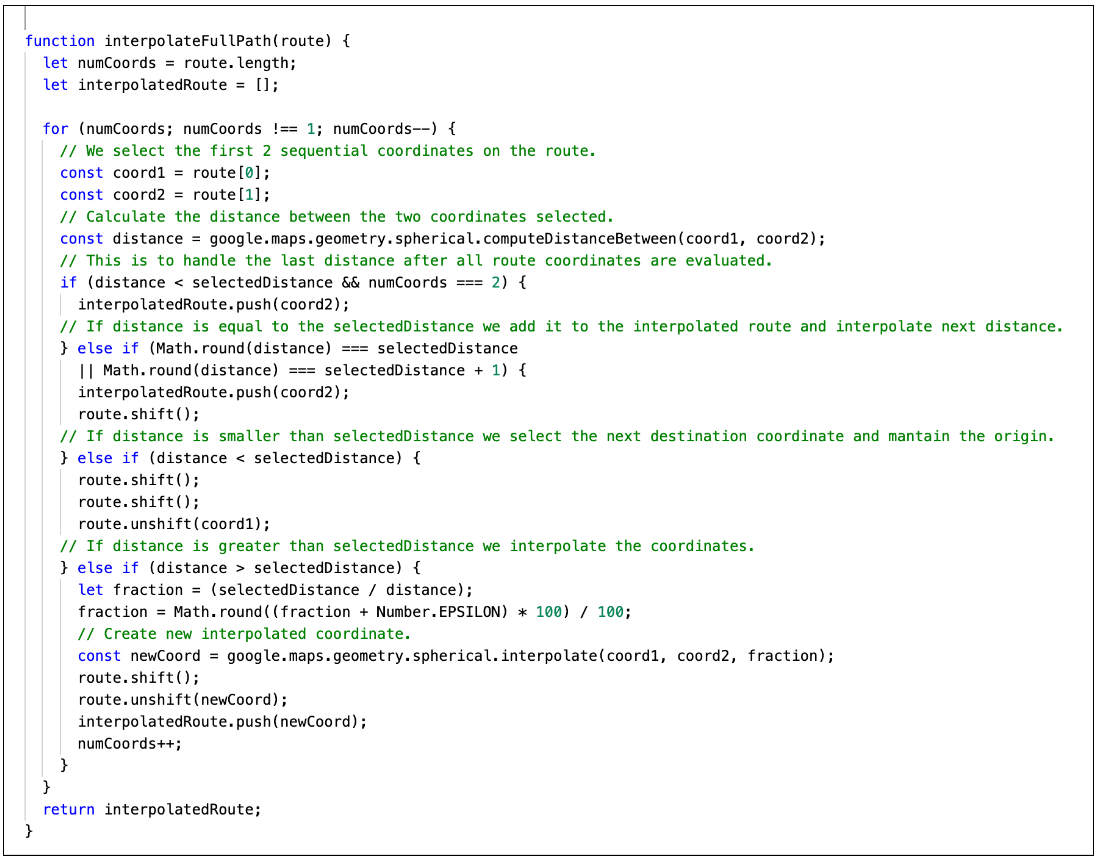


Figure 4.3: Termite2 GUI interface implementation.

Termite2 GUI performs two types of communications with the Termite2 Client: i) requesting data, for example, requesting emulators to be used on the emulated network; and ii) or sending data, for example, sending commit messages to the target emulators or commands to start new emulators.

These communications are implemented using HTTP GET and POST requests and the data is formatted using JSON objects. These requests are sent from the web browser to a web service (implemented using Java Servlets running inside the Tomcat Server) that translates them to Termite2 Commands (similar to those used when the user uses the console interface). The commands are then sent from the Servlets to the Termite2 Client (through a socket connection on localhost:8081) to be processed. Fig. 4.3 shows these communications.



```javascript
function interpolateFullPath(route) {
  let numCoords = route.length;
  let interpolatedRoute = [];

  for (numCoords; numCoords !== 1; numCoords--) {
    // We select the first 2 sequential coordinates on the route.
    const coord1 = route[0];
    const coord2 = route[1];
    // Calculate the distance between the two coordinates selected.
    const distance = google.maps.geometry.spherical.computeDistanceBetween(coord1, coord2);
    // This is to handle the last distance after all route coordinates are evaluated.
    if (distance < selectedDistance && numCoords === 2) {
      interpolatedRoute.push(coord2);
    // If distance is equal to the selectedDistance we add it to the interpolated route and interpolate next distance.
    } else if (Math.round(distance) === selectedDistance
      || Math.round(distance) === selectedDistance + 1) {
      interpolatedRoute.push(coord2);
      route.shift();
    // If distance is smaller than selectedDistance we select the next destination coordinate and mantain the origin.
    } else if (distance < selectedDistance) {
      route.shift();
      route.shift();
      route.unshift(coord1);
    // If distance is greater than selectedDistance we interpolate the coordinates.
    } else if (distance > selectedDistance) {
      let fraction = (selectedDistance / distance);
      fraction = Math.round((fraction + Number.EPSILON) * 100) / 100;
      // Create new interpolated coordinate.
      const newCoord = google.maps.geometry.spherical.interpolate(coord1, coord2, fraction);
      route.shift();
      route.unshift(newCoord);
      interpolatedRoute.push(newCoord);
      numCoords++;
    }
  }
  return interpolatedRoute;
}
```

Figure 4.4: Interpolation function.

The new movement option mentioned above, allows the user to select a desired destination for a network node; then, Termite2 creates a route and draws it on the map displayed in the user interface in which the node moves at a fixed speed, chosen by the user; the speed at which a node moves emulates walking (5m/s), bicycling (10m/s) or driving (20m/s).

To create the movement route, Termite2 starts by making an HTTP request to the Google Maps API Directions service using JavaScript. The request contains the origin coordinates of the selected node, the destination coordinates and the travel mode (the travel mode can be walking, bicycling or driving, and it indicates what traffic rules and lanes should be used when creating the route on the map being used). The request response returns an array of coordinates that delineates a path between the origin coordinates and the destination (we call this array, the route array).

However, the coordinates obtained on the route array are not equally spaced between each other, i.e., the distances between each sequential coordinate is not fixed. In fact, the first two coordinates can be at a distance of 20 meters while the distance between the next two can be 40 meters. This means that the coordinates obtained can not be immediately used to realistically emulate the movement of the nodes. Note that moving a node one coordinate per second does not work if the distance between sequential coordinates does not correspond to a fixed value that emulates walking, bicycle or driving speed. To solve this problem we interpolate the coordinate values on the route array and create a new array of interpolated coordinates that (in sequence) are equally distant from each other. The distance used is selected (on the configuration menu on the interface) by the user and is related to the following speeds 5m/s, 10m/s and 20 m/s. Fig. 4.4 shows the JavaScript function responsible for performing the interpolations here described (the comments on the code explain its logic). The interpolated route is then drawn on to the emulated network and when the user starts the automatic move event the node moves along the interpolated route one coordinate each second. This emulates the movements previously presented: walking (5m/s), bicycling (10m/s) or driving (20m/s).

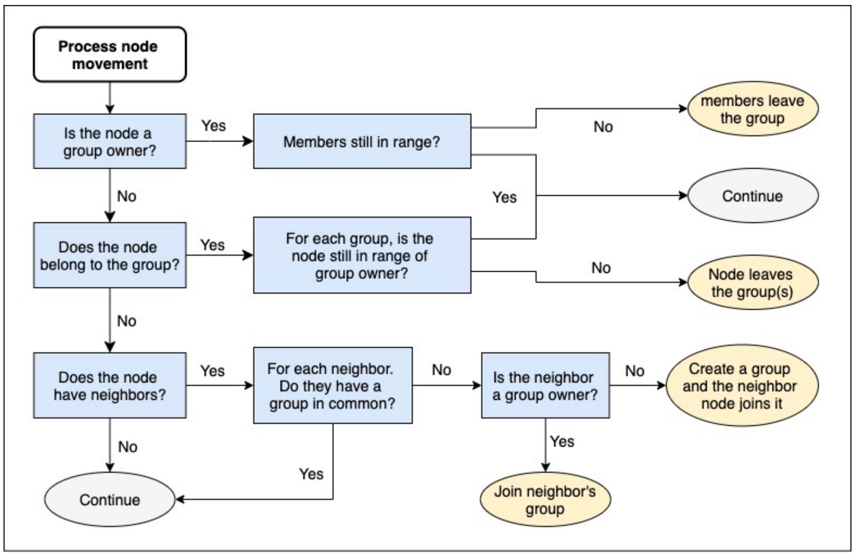

Process node movement
Is the node a group owner?
Yes
Members still in range?
No
members leave the group
Yes
Continue
No
Does the node belong to the group?
Yes
For each group, is the node still in range of group owner?
No
Node leaves the group(s)
No
Does the node have neighbors?
Yes
For each neighbor. Do they have a group in common?
No
Is the neighbor a group owner?
No
Create a group and the neighbor node joins it
Yes
Join neighbor's group
No
Yes
Continue

Figure 4.5: Automatic peer-to-peer events logic.

It is important to note that the walking speed of 5m/s is not very accurate (the average walking speed is around 2m/s). We chose the 5m/s speed value due to the fact that if we try to interpolate the coordinates on the route array (for walking speed) to distance values smaller than 5 meters, we obtain the same coordinate values as the origin or the destination of the interpolation. Therefore, 5 meters is the smallest interpolation distance between two coordinates that we can perform. Nevertheless, when we consider the automatic movement option, the speed at which a node moves is merely representative;

the most important aspect is that the node movement speed is consistent in order to emulate realistic movement.

Finally, it is important to present the logic that we use to generate the automatic peer-to-peer group interactions between nodes when these move using the automatic move event (see, Fig 4.5). When the nodes move along the interpolated route they automatically create, join and leave peer-to-peer groups with other nodes. This happens when nodes get in range of each other. We remember that the automatic node movement and peer-to-peer interactions between the nodes is not transited to the target emulators in real time. What we mean by this is that (for example) when a group is created between two moving nodes the associated emulators (target emulators) do not immediately receive a commit messages that indicates to each one that a peer-to-peer group was created between them. Instead, we first generate the automatic movement event and then commit all the peer-to-peer interaction that happened during the node(s) movement. This happens due to the single threaded nature of JavaScript, which does not allow us to process the movement of the nodes and send the peer-to-peer events generated (commits) to the target emulators at the same time, while maintaining the nodes movement consistent (without pauses) and the commits synchronized (without delays). A possible solution to solve the single threaded nature of JavaScript would be to use a JavaScript library with multi-thread capabilities like the Node.js library.

4.3 Summary

This Section presented the implementation of Termite2's new components (when compared with the old system version, Termite). The Termite2 Server was implemented using Java, this guarantees that the system can easily run on any common operating system like, Windows, Linux or MacOS and at the same time stays consistent with the rest of Termite code base (which is already in Java). When we look at the Java classes that implement the Termite2 Server the most important ones to consider are those that handle communication from and to the Termite2 Server (ClientReceiverThread, LocalReceiverThread and ServerReceiverThread classes), and the AVDControllerDriver class witch contains all the necessary methods to interact and configure the emulators. Finally, we discussed how we implement some of the most important components in Termite2 GUI. The new interface was implemented using JavaScript and it leverages the Google Maps API to display the emulated network on a real world map. We also presented and explained the logic used to implement the new automatic movement option provided by the new interface.

5

Evaluation

Contents

Features	Termite Console	Termite2 GUI
Create virtual network nodes	X	X
Emulate node movement	X	X
Create peer-to-peer groups	X	X
Automatic node movement and interactions	-	X
Manage multiple emulators	-	X
Manual node binding to emulators	X	X
Automatic node binding to emulators	-	X
Emulated network displayed on a world map	-	X
Emulation of small networks	X	X
Emulation of large networks	X	X
Perform simple tests (small number of node interactions)	X	X
Perform complex tests (large number of node interactions)	-	X
Save/load network typologies	X	X
Save/load test scenarios	X	X
Step by step execution	X	X

Table 5.1: Termite2 GUI and Termite console features

In this chapter we present Termite2 evaluation. The tests evaluate Termite2's main requirements: usability, performance, and scalability. All tests were performed both on Termite2 and Termite in order to see how Termite2 succeeds in providing usability and scalability improvements over Termite, while maintaining acceptable levels of system performance.

5.1 Usability

When compared to Termite, Termite2 improves the system usability by providing a graphical user interface from which developers are able to develop and test their applications in a quick and easy way. Thus, evaluating the system usability of Termite2 against Termite means that we are evaluating Termite2 GUI against Termite console interface.

In Chapter 1.2 we defined the requirements Termite2 interface must have:

- be a graphical interface;
- have a clear and useful presentation;
- allow developers to easily see and model their emulated networks;
- be able to visualize the various virtual nodes and their movement/interactions within the network;
- the network should be displayed on a real world map in order to ground the emulated network and interactions within;
- able to create small and large networks;

- help in the development of both simple and complex applications and tests that properly emulate real world peer-to-peer scenarios;
- from this interface we must able to deploy the necessary emulators instances and the desired Android applications.

These requirements can be translated to features that Termite2 interface must have. On the Table 5.1, we present these features and show how Termite2 GUI compares against Termite console interface. Nevertheless, to properly evaluate all requirements (for example: how clear, useful and easy is the new interface) without a bias, we need to perform proper usability tests with real users, and compare the usability of Termite2 GUI against Termite's console interface.

5.1.1 Tests

To evaluate and compare the usability of both Termite and Termite2 interfaces we developed a tests guide ([49]).The goal is to evaluate how users perform two test cases. These tests were first done with Termite (using the console interface) and then with Termite2 (using the new GUI). We started each test with a brief presentation on how each system works and what their features are. All tests were performed with one user at a time and followed a lab testing approach [50] with both unmoderated and moderated approaches. We chose this form of testing because it provides in-depth information on how the user interacts and feels about the system.

All tests were performed on a node using Windows 10, equipped with an Intel i5 6500 CPU, and 8GB of RAM. The emulator instances used corresponded to a Pixel2 Android phone running Android 5.1 (API 21) with 1GB of RAM. Both Termite, Termite2 and the emulators used, were already installed on the Windows node before the tests. The users performed the tests remotely from their homes. To do this we used Google Chrome Remote Desktop [51] to give users remote access to the test machine and the systems being tested.

After the tests were concluded we asked each user to answer a set of questions ([49]) that allowed us to gather data about their experiences. With this data we were able to evaluate how each user experienced both interfaces and allowed us to compare the usability of Termite2 against Termite.

We performed the tests with five different male users between the ages of 20 and 25. All users were experienced with software and had previously used Termite for academic projects.

It is important to note that although five users is a very small sample size for our usability test, we need to take into consideration that this project was developed during the COVID-19 pandemic. As such, the realization of these tests had to be carried out exclusively remotely, which made it very difficult to gather a large number of users to perform the tests.

Nevertheless, given the profile of the five users that performed the tests (previous Termite users) we think that the results obtained still allow us to take valuable conclusions over the usability of both systems and their interfaces.

5.1.2 Results

As we have stated before, Termite2 should present a clear usability improvement over the Termite system. After analysing both systems interfaces we can clearly conclude that Termite2 GUI is an obvious improvement over Termite console interface. We can all agree that a visual interface is almost always better than a non-visual interface. This is specially true when we consider the nature of both Termite2 and Termite systems and the emulated network they create. In fact, by allowing users to see the network being emulated on a real world map and all the possible interactions that can happen within, Termite2 helps to ground the network and the peer-to-peer scenarios being emulated.

When using new features such as automatic node binding, management of multiple emulators instances from the interface, automatic movement of multiple nodes, and peer-to-peer interactions, a user can create large networks with complex peer-to-peer scenarios. Note that Termite2 also maintains all Termite's interface features done through a console but translates them to interactive button clicks and selections on a graphical interface.

Lastly, our usability tests results (can be found at [49]) show that all users prefer the new graphical interface over Termite console in all aspects. One could of course argue that with only five users, the results obtained are not sufficiently conclusive (which is correct). However, we feel that the visual nature of the new interface and its features clearly present an evolution over Termite console usability and that all usability requirements presented in Chapter 1.2 were fulfilled.

5.2 Performance

In Chapter 1.2 we presented Termite2 requirements; in particular, the performance of Termite2 must not be worse than Termite´s. When we look to both Termite and Termite2's architectures we can see that the main differences, besides the new interface layer, is the new Termite Server component in Termite2. This component completely changes how messages are exchanged within the system: both when we consider the communication protocols between Termite Client and the emulators, and also between emulators. In fact, note that in Termite messages are exchanged directly from the sender to the receiver. On Termite2 (due to Termite Server component) such communication is not direct because Termite Server acts has a middle point responsible for redirecting messages sent from Termite2 Client to the target emulators and messages exchanged between emulators.

Thus, evaluating Termite2 performance against Termite consists mainly in evaluating the impact that Termite Server has on such communication. To evaluate such communication, we developed two tests:

i) the Ping Test (see Section 5.2.2) measures the time that it takes to send a ping message from the Termite Client to the emulators and receive a response and do the same using the Termite2 Client; ii) the Share Test (see Section 5.2.3) measures the time it takes to send a message from one emulator to another and receive a response.

Furthermore, Termite2 allows us to run emulators instances distributed throughout multiple remote machines (which Termite does not). For this reason the two performance tests that we present (Ping and Share) were performed in two parts. First, we run both tests on Termite and Termite2, with both systems running locally (i.e., all system components and emulators run on the same developer machine). Then, we execute the same tests on Termite2, this time running all the necessary emulator instances remotely. This means that on one machine we run the Termite2 Client while the necessary emulators run in multiple remote machines. Obviously, on each remote machine there is as instance of Termite2 Server. The number of emulators running on each machine vary according to the test being performed. With this approach we ensure that the performance evaluation of Termite2 against Termite is fair, while also evaluating, in the case of Termite2, the performance impact of running the emulator instances locally or remotely.

5.2.1 Testing environment

This section describes the testing environment used on both performance tests (Ping and Share). As stated before, both tests were done in two distinct environments: i) all emulators and system components run in a single machine (i.e., Termite2 being used as if it was Termite), and ii) emulators run distributed through multiple remote machines (which is possible only with Termite2).

When performing the tests locally, we executed all elements of both Termite and Termite2 on a single Rede das “Novas” Licenciaturas (RNL) node.[1] Each RNL machine runs Ubuntu 18.04, with an Intel Core i5-4460 3.20GHz (Quad-core) CPU, with 16GB RAM (around 14GB usable) and a network speed of 250 Mb/s. All the emulators used correspond to a Pixel2 Android phone running Android 5.1 with API 21. We use this API version because Termite API was developed for this Android version; so, for Termite2 we used the same API.

When performing tests where the emulators instances are distributed throughout different remote machines, the Termite2 Client was executed in a MacBook Pro running macOS Catalina 10.15.7, featuring a (Dual-Core) Intel Core i5 2,7 GHz CPU (4 threads), with 8GB of RAM and a network speed of 80 Mb/s. This machine was operating inside a home network outside IST network environment, while all the emulators were distributed (according to the test) throughout multiple RNL computers inside IST network. These machines are situated at IST on laboratory 13 in the Alameda Campus.

As shown in Section 3.3.1, Termite2 Client can only connect with Termite2 Servers that are running

[1] RNL nodes/machines are computers available at Instituto Superior Técnico (IST) laboratory 13

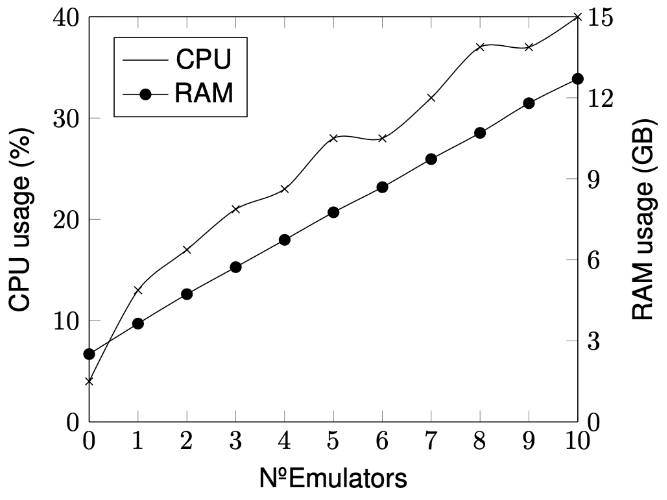


Figure 5.1: CPU and RAM usage with an increasing number of Android emulators.

on machines within the same local network. However, this can be bypassed if we use software that allows Termite2 Client to see the remote networks of the machines as local ones. As such, to connect the Termite2 Client running on our home machine (on a home network) to the Termite2 Server(s) on the remote RNL machines (on IST network) we used the sshuttle software [47].

With sshuttle we could forward the IP addresses and ports of an entire remote network (in our case the lab 13 network at IST on the RNL machines) to a local network (our home network). In short, this software allows the local network of each RNL machine to be accessed directly from our home machine as if they were all in the same network. For a more detailed explanation on how sshuttle works please see the official documentation [47].

Lastly, it is also important to note that when running emulators on the RNL machines, we never exceeded the maximum number of ten emulators instances running at the same time on the same machine. This was done due to memory constrains on each RNL machine. Android emulators can be extremely heavy specially on system RAM. In Fig. 5.1 we show how CPU and RAM usage on a computer are affected when increasing the number of emulator instances. Running 10 emulators consumes around 13GB of RAM with a 40% of CPU usage. With the RNL machines having only 16GB of RAM (14GB usable), running more then ten emulators starts to severely impact system speed and responsiveness. As such, we consider ten emulators to be the maximum number of emulator that we can run on a single RNL machine.

5.2.2 Ping Test

This test was developed to evaluate Termite2 Server impact on the performance of Termite2 when messages are exchanged between Termite2 Client and the emulators.

The test consists in measuring the time it takes to send a ping message from the Termite2 Client (and then on Termite Client) component to an increasing number of emulator instances. Thus, we created a test script that would send multiple ping message to the emulator(s) and measured the time that each ping took to be received and a response being sent. This script can be seen at [49]. Starting with one emulator instance, we executed the test script until the variance of time values obtained becomes less than 10%. We then increased the number of emulators and performed the same test until a maximum of ten emulators instances was reached. After all tests were concluded, we calculated the average value of each test.

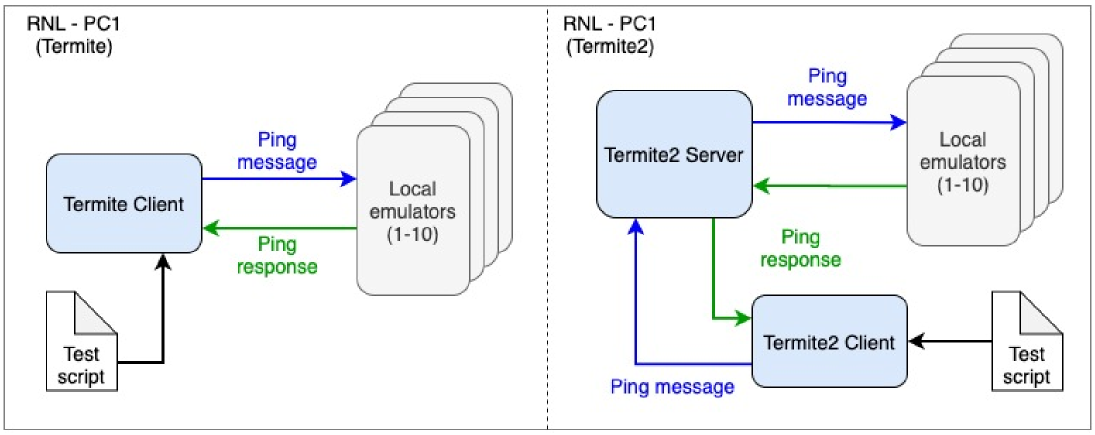


Figure 5.2: Ping test on Termite and Termite2 with emulator running locally

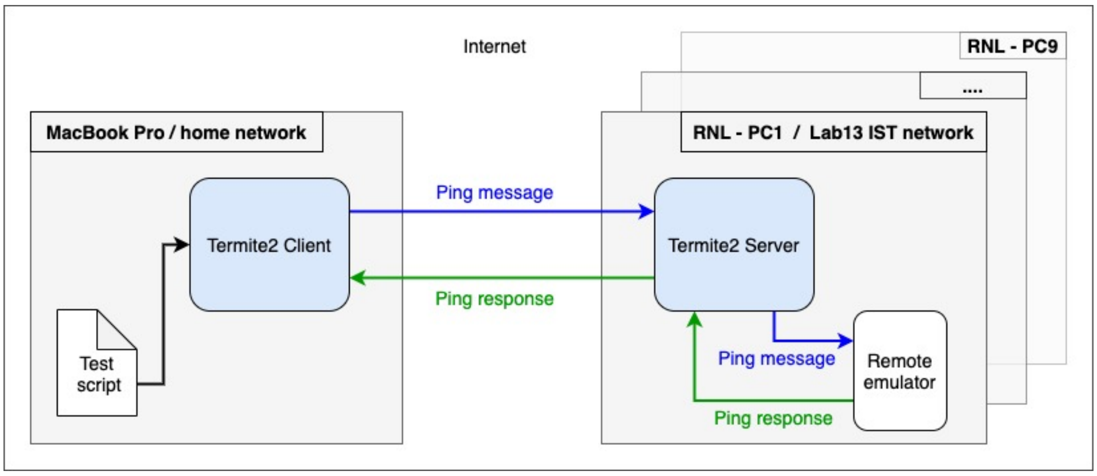


Figure 5.3: Ping test on Termite2 with emulators distributed across different machines

As explained before, the test was performed in two different scenarios. First, we executed the tests running all system components and the emulators locally (see Fig.5.2, Termite on the right and Termite2 on the left). Then, we performed the same test but instead of increasing the number of emulators that run locally, we increased the number of remote (RNL) machines being used with each machine running a single emulator instance (see Fig.5.3).

5.2.2.A Results

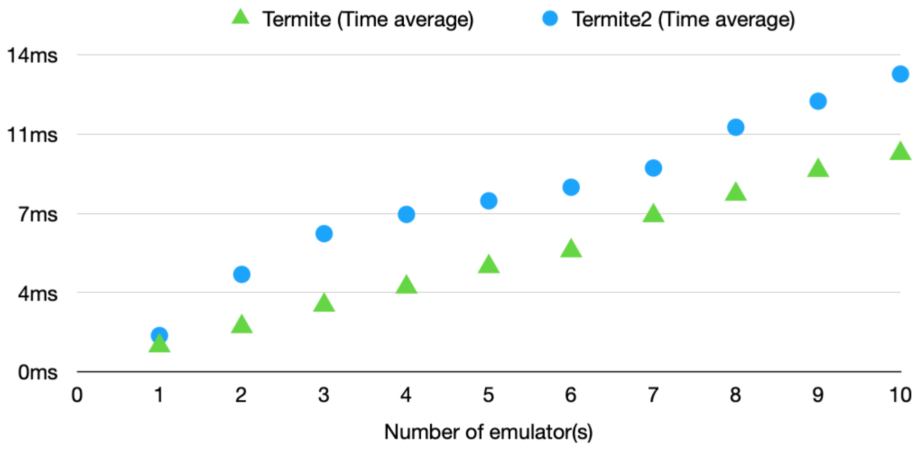


Figure 5.4: Ping test local results

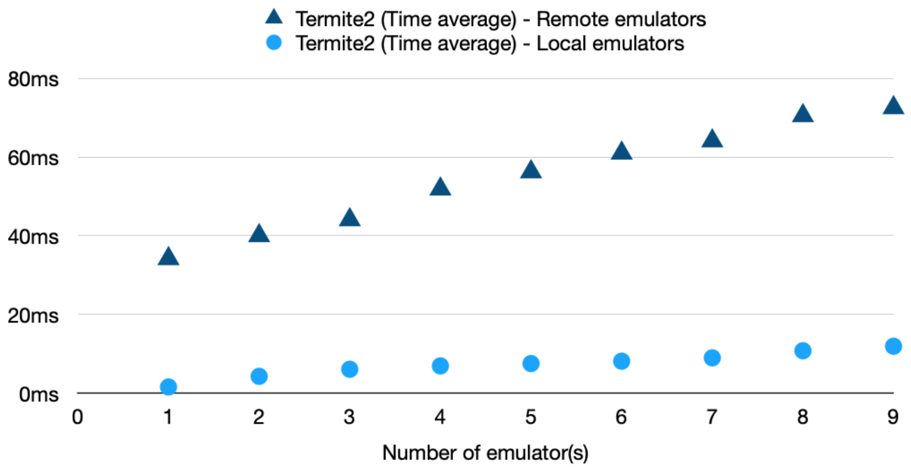


Figure 5.5: Ping test distributed emulators results

The results obtained when performing the Ping test locally on Termite and Termite2 can be seen in Fig.5.4. Results show that Termite2 Server has a negligible impact on the time that it takes to send the ping messages (less than 3 milliseconds). This allows us to conclude that Termite2 presents a performance similar to Termite's on this type of communication within a local environment.

For the tests conducted with each emulator running in a different remote RNL machine on laboratory 13, the results show (see Fig.5.5) there is a performance impact when compared with the local Termite2 test results (around 40 to 70 milliseconds depending on the number of remote machines). The reason for this increase is that messages are exchanged over the Internet between Termite2 Client (on a home network) and the RNL machines on the IST network. This is expected and we believe that 40-70 ms of latency does not compromise usability of the system (many online games have similar latencies). Therefore, when we consider the communication between the Termite2 Client component and the emulator instances we believe that the time values obtained are still within very reasonable values.

5.2.3 File Sharing Test

This test evaluates Termite2 Server impact on the performance for exchanging messages between two emulators. The test consists in measuring the bandwidth between two emulators by measuring the time that takes to send a varying size file (1 Mb to 10 Mb) from one emulator to another when they are part of a peer-to-peer group. Thus, we created an Android mobile application (running in an emulator) that detects the creation of the peer-to-peer group and automatically sends the file to the other group member (i.e., the other emulator). The application would then measure the time between sending the file and receiving a response from the other group member. In order to trigger this peer-to-peer event between the two emulators, we created a script using Termite2 commands [49]. that emulated the creation of the peer-to-peer group between two emulators running the app previously mentioned. We executed this test script until the variance of time values obtained was less than 10%.

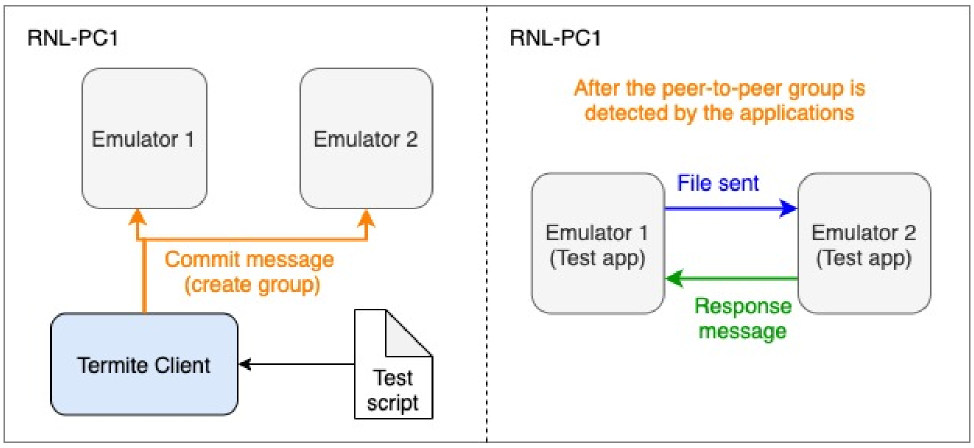


Figure 5.6: File Sharing test on Termite with emulators running locally

Similar to the Ping Test, this test was performed in two different scenarios. First we executed the

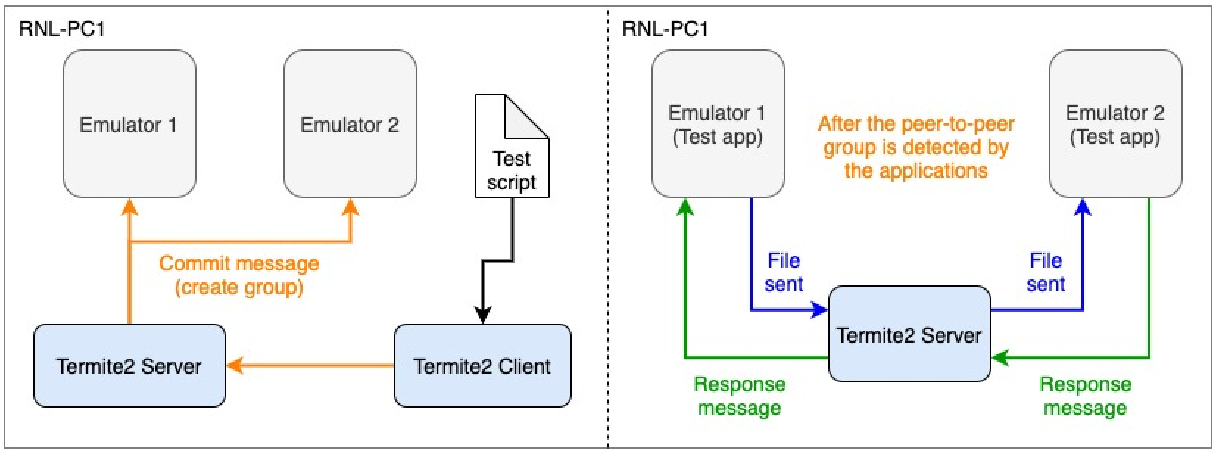


Figure 5.7: File Sharing test on Termite2 emulators running locally

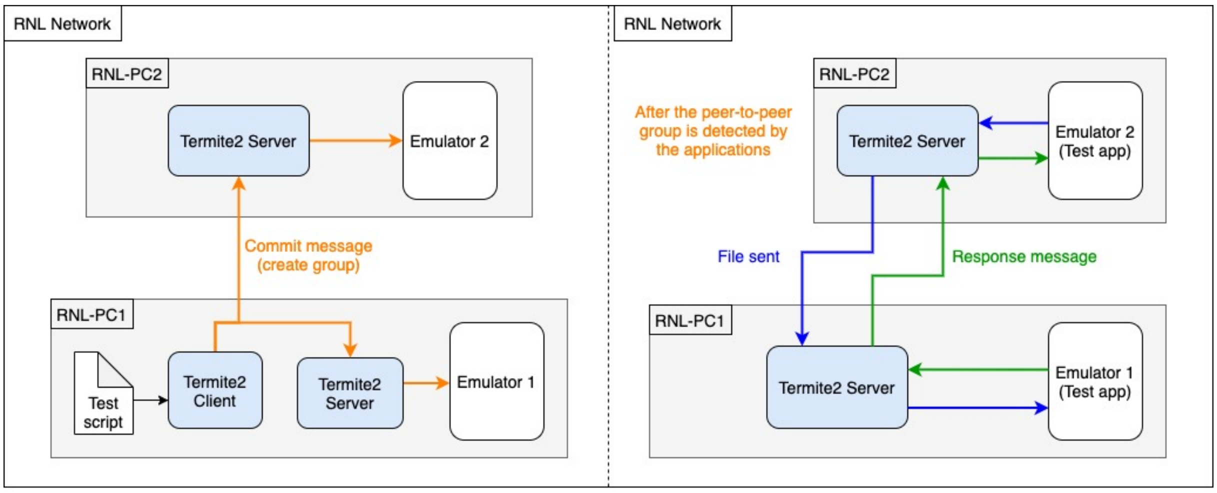


Figure 5.8: File Sharing test on Termite2 with the emulators running on different machines

tests running both emulators on the same machine (RNL-PC1). In Fig.5.6 and Fig.5.7, we show how the test runs. We first set the emulated group with a commit message sent to the emulators (left side of the images), which then triggers their automatic communication (right side of the images). Finally, we performed the same tests, this time running one emulator on RNL-PC1 and the other emulator on RNL-PC2. In Fig.5.8, we show how the test runs. We first set the emulated group with a commit message sent to the emulators (left side of the images), which then triggers their automatic communication (right side of the images). After all tests were finished, we calculated the average time value for each test. The results are shown and discussed in the next section.

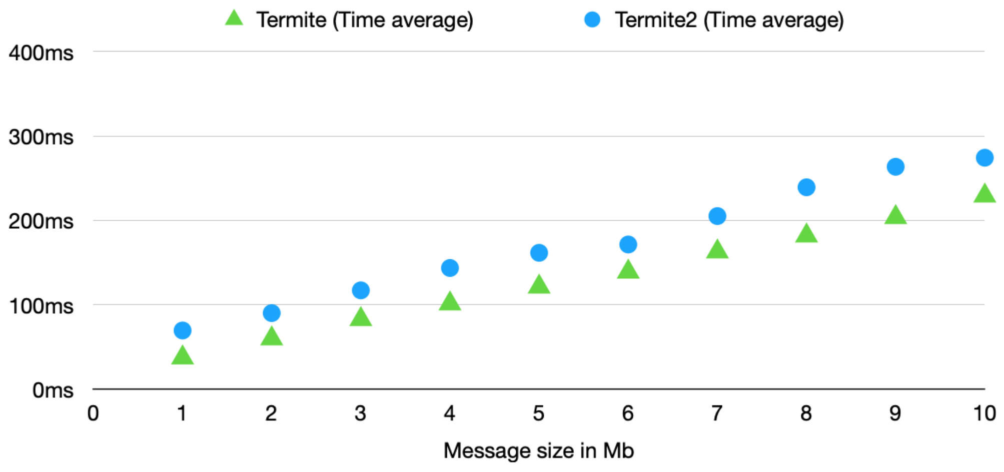


Figure 5.9: File Sharing test local results

5.2.3.A Results

The results obtained when performing the File Sharing test on Termite and Termite2 with both emulators running on the same machine can be seen in Fig.5.9. The results show that Termite2 Server has a negligible impact on the time that messages take to be sent from one emulator to another (an increase of around 30 milliseconds per megabit). So, we conclude that Termite2 presents a similar performance as Termite on this type of communication when emulators are running on the same machine.

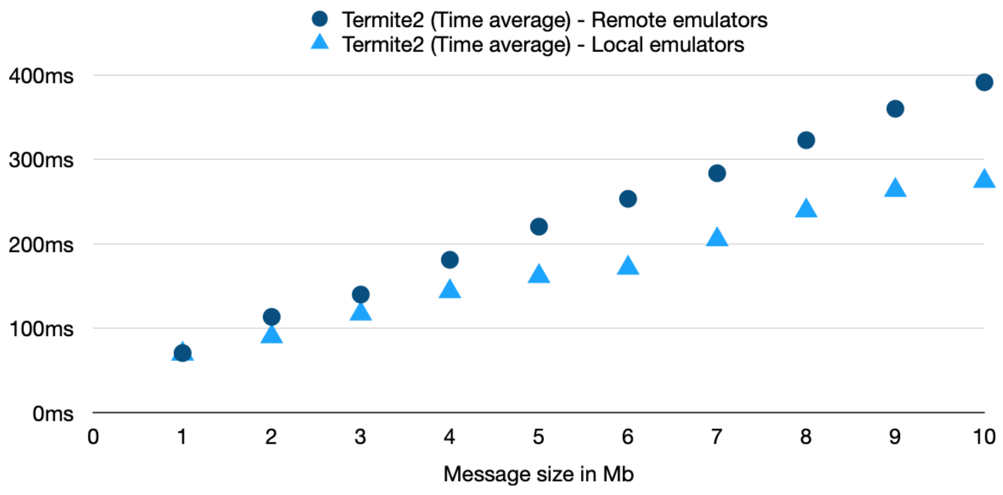


Figure 5.10: File Sharing test remote results

For the tests conducted on Termite2 with each emulator instance running on different RNL machines, the results show (see Fig.5.10) that there is a performance impact when compared with the local Termite2 test results (and increase of 20-100 milliseconds per megabit when compared with the results obtained on Termite2 using local emulators). The reason for this time increase is the fact that the messages sent from one emulator to another are done from two different RNL machines over the same network. Figures 5.7 and 5.8, show the difference between the travel path of the messages when the emulators are running on the same machine or distributed throughout different ones. Nevertheless, when we consider the applications that we are trying to develop and the network paradigm being used (encounter networks), sending large files through a peer-to-peer connections (WiFi-Direct), is not recommended due to the unstable nature of this type of communication (devices come in and out of proximity of each other).

In conclusion, Termite2 Server has a negligible impact on the time that an emulator takes to send and receive a message from and to another. This happens due to the fact that messages are not sent directly between them (as is the case when using Termite). Instead, messages are first received by the local Termite2 Server and redirected to the destination (to the Termite2 Server that is managing the target emulator). When both emulators are running on the same machine, the performance impact is small and greater if the emulator is running on a different machine. However, we feel that in both cases the performance impact is acceptable for the communication paradigm we are using (encounter networks) and we believe that Termite2 presents an acceptable performance difference when compared to Termite.

5.3 Scalability Tests

Besides usability, improved system scalability is another of Termite2's main requirements. System scalability is directly related with the number of emulator instances we can use within the emulated network running the application we wish to develop and test. Thus, for Termite2 to be more scalable than Termite, it needs to be able to properly support a larger number of emulated Android devices.

We have previously mentioned that when using Termite we need to use the Android Studio AVD Manager or user made scripts to create and manage the emulator instances. We also know that due to Android SDK limitations we can only start a maximum number of 16 emulators instances at the same time on a single machine. As such, when using Termite we are only able to create an emulated network with a maximum of 16 target emulators. This number assumes that the local machine has all the resources needed for that purpose, which is not realistic for most commodity laptops and desktops.

With Termite2 we can now create and manage all emulator instances from the Termite2 Client itself. This removes the need to use Android AVD Manager or user made scripts to manage the emulators. Using Termite2 and the new Termite2 Server component we also provide a solution to the Android SDK limitation on the maximum number of emulators we can run locally. This is done by allowing a user to

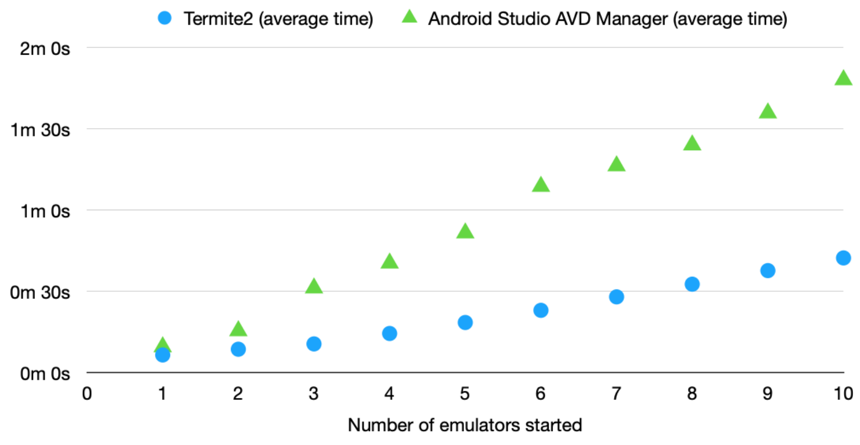


Figure 5.11: Local Deployment results for Termite and Termite2

create the emulated network on one machine and run the emulator instances distributed across other machines, allowing the user to create much larger emulated networks.

Nevertheless, we cannot present these Termite2 features as an immediate justification for the system improved scalability when compared to Termite. For Termite2 to be truly scalable it needs to be able to deploy a large number of emulators within an acceptable period of time. To evaluate this we developed the tests described in the following sections.

5.3.1 Local Deployment

This test evaluates the time that it takes to launch an increasing size of emulator instances running a test application using both Termite and Termite2. The test consists of measuring the time it takes to launch a selected number of emulator instances, installing an applications on each emulator and run it.

On Termite we must use Android Studio AVD Manager. Using this Android Studio utility the test corresponds to selecting the desired number of emulators to start and the application to run. To perform this test on Termite2 we created a simple script file [49] using Termite2 commands to perform the same actions of launching the emulators, install the applications and run them. Starting with one emulator instance to a maximum of ten (the maximum is 10 emulators for reason already explain on Section 5.2.1), we executed this test until the variance between time values obtained was less than 10%. This test was performed on a single RNL machine (RNL-PC1).

The results obtained for this test are shown in Fig.5.11 (note that the Termite results correspond to the values obtained when using Android Studio AVD Manager for reasons already explained). By looking at the graph we can clearly see that Termite2 presents a significant improvement on the time that it takes to launch the emulators, install the applications and start them. The reason for this improvement is due to the fact that Android Studio AVD Manager starts the emulator instances in sequence (one after another) while Termite2 starts all emulator instances at the same time. We can see this by considering that the time improvement when we use Termite2 is greater when the number of emulators used increases.

5.3.2 Distributed Deployment

From the previous test results we were able to show how Termite2 speeds up the deployment of increasing number of emulators and applications on a single machine. However, because we are running all the emulator instances on a single machine we are still limited by its performance and the hard limit imposed by Android SDK of 16 emulators instances running at the same time.

To truly show how Termite2 improves system scalability we developed a test similar to the previous one but now each emulator instance runs on a unique RNL machine. To perform the test we used a similar script file as the one created for the Local Deployment test [49], but now we distribute the emulators across multiple RNL machines. This script file was loaded on Termite2 Client running in a home machine (on a home network) and the emulators started on remote RNL machines (on IST network).

First, we started by measuring the time it takes to deploy a single emulator per machine, installing the template application and start it. With only 9 RNL machines available, we were able to reach a total number of 9 emulators, which then allows us to compare these results to those obtained on the Local Deployment test (with Termite2 on a single machine).

Finally, we perform the same test but this time we deployed 10 emulators instances per remote RNL machine. As explained before we stopped at ten emulators due to performance constrains on the RNL machines. With 9 RNL machines available we were able to launch a total of 90 emulators, a drastic improvement over the maximum 16 emulators that one can run when using Termite.

In Fig.5.12 we show the values obtained when starting one emulator across the 9 available RNL machines and compare these values to those obtained on the Local Deployment test, where we started the same amount of emulators (using Termite2) on a single RNL machine. We can see that with the distributed approach the time it takes to start one emulator on a single machine and starting 9 emulators across 9 different machines is approximately the same. This happens due to the fact that the Termite2 Client processes/sends the start commands to the Termite2 Servers on multiple machines at the same time. Thus, with Termite2 it is possible to explore the inherent parallelism of a distributed system

This parallel processing is again shown on the results obtained when starting ten emulators across the nine RNL machines (see Fig.5.13). As expected, starting 10 emulators on a single machine takes approximately the same time as starting 90 emulators across 9 different machines (with 10 emulators per machine).

Finally, by looking at Fig.5.13 and comparing the results obtained to those that we presented on the

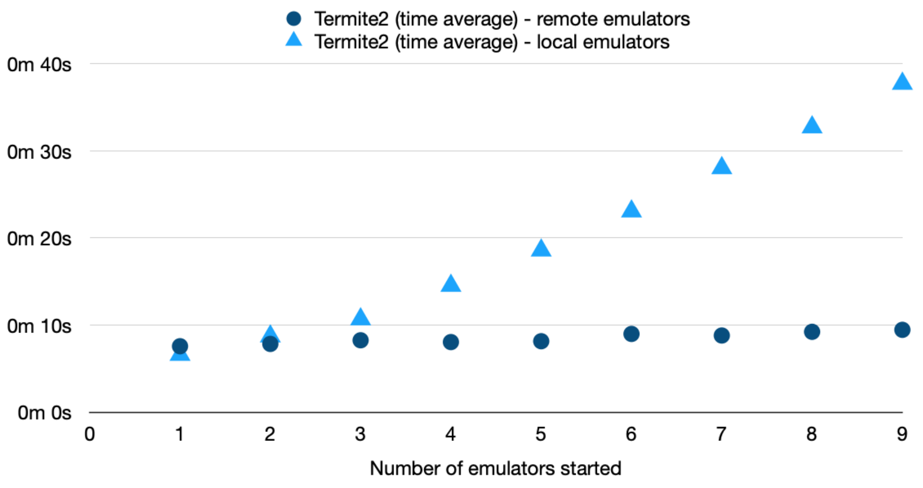


Figure 5.12: Startup time for local and remote emulators using Termite2

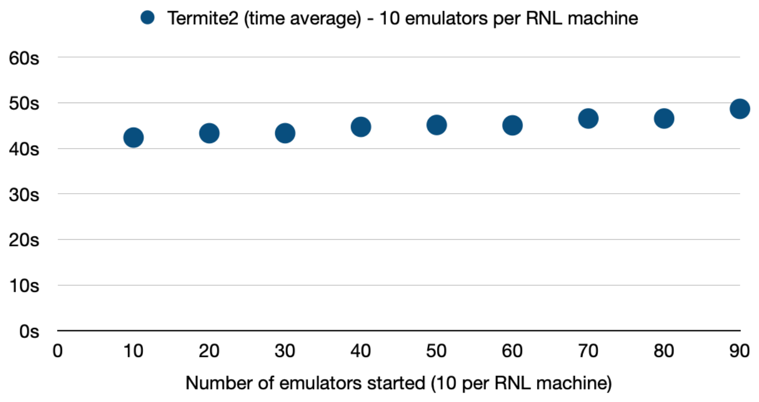
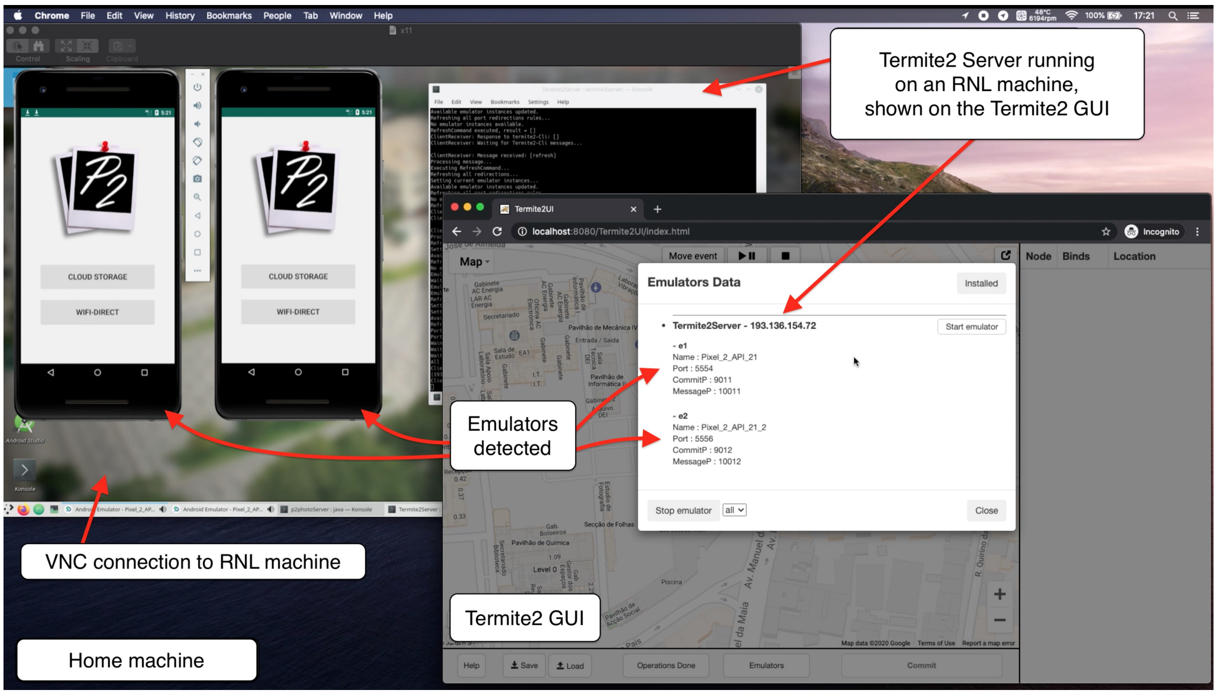


Figure 5.13: Maximum number of emulators we are able to use with 9 RNL machines

Local Deployment test we can see that starting 90 emulators takes less time than to start 10 emulators on Termite (using Android Studio AVD Manager, which starts the emulators sequentially).

With these results we can easily conclude that Termite2 is much more scalable than Termite.



```
Termite2Server - 193.136.154.72
- e1
Name : Pixel_2_API_21
Port : 5554
CommitP : 9011
MessageP : 10011
- e2
Name : Pixel_2_API_21_2
Port : 5556
CommitP : 9012
MessageP : 10012
```

Figure 5.14: Using Termite2 to test P2photo application.

5.4 Use case

As a final test for the Termite2 system, we show in detail a practical example on how Termite2 can be used to support the development/test of an encounter application, while highlighting Termite2's new features. Note that we omit how to install Termite2 and its components; to learn how this is done please see the appendix at the end of this document A.

For this test case we use an Android application called P2photo. P2photo was developed for a project within the scope of the Mobile and Ubiquitous Computing course at IST [52]; it allows users to create photo albums and share them among other P2photo users using WiFi-Direct. The application implements the Termite2 API. Thus, to test the peer-to-peer capabilities of P2photo we use Termite2 to create a test scenario where two P2photo users come into range of another and, if the application behaves as expected, a photo is automatically shared between them.

We now explain how we deployed Termite2 and the emulators used, to create this test scenario. All the machines used for this test have the same configurations as the ones that we present in Section 5.2.1.

We start by establishing a Virtual Network Computing (VNC) connection to a RNL machine (in IST network, the RNL machine local network IP was 193.136.154.72) on which we installed and started a Termite2 Server. In our home machine (inside a home network) we installed Termite2 Client. To connect the Termite2 Client to the Termite2 Server running on the RNL machine we registered the RNL machine local network IP (193.136.154.72) on Termite2 Client and used sshuttle [47] to allow the bypassing of Termite2's local network requirements (for reasons already explained in Section 3.3.1). We then started the Termite2 Client; on startup, the Termite2 Client automatically connects to the registered Termite2 Server(s) and prompts the user to select the desired interface option (the console interface or the Termite2 GUI). For this use case we choose the Termite2 GUI option accessed through a Chrome browser (we could have chosen other web browser) window on `localhost:8080/termite2UI/index.html`.

Using the Termite2 GUI we then started two emulators on the RNL machines running the P2photo. Fig. 5.14 shows the current state of our deployment after the emulators startup. From this point on, we emulate the peer-to-peer scenario where two P2photo users get close to each other and a photo is automatically shared between them. To do this, we simply need to: i) create two virtual nodes on the emulated network view (by right clicking on the map and choosing the `create node` option); ii) bind the nodes to the emulators; iii) move one of the nodes into the proximity of the other; iv) create a peer-to-peer group between the two nodes (by left clicking on one of the nodes and selecting the `create group` option); and v) press the commit button.

After the commit button is pressed, commit messages are sent to the target emulators (the emulators bound to the nodes) which makes the P2photo react accordingly to the emulated scenario, and a photo is exchanged between both devices.

5.5 Summary

In this chapter we present the evaluation done on Termite2 in order to see if the system satisfies the goal and requirements that were indicated in Section 1.2. We compared the Usability, Performance and Scalability of both Termite and Termite2, shown the performance results of Termite2, and presented a real use case using Termite2 to develop a simple peer-to-peer scenario on the P2photo Android application (project used in 2019 at IST in the course named Mobile and Ubiquitous Computing).

The usability tests performed on Termite2 show that the new GUI improves Termite2's system usability when compared to the current Termite Console interface option. Termite2's new interface option is not only easier to use but through its visual nature is able to help users to better understand peer-to-peer scenarios and the network paradigm (encounters network) they are trying to emulate. Termite2 shows some performance overhead when compared to Termite (due to messages being redirected through the Termite2 Server component); this is true for emulators running on the same machine as the Termite2 Client component or distributed across multiple machines. However, we believe that the Termite2 results are still within acceptable values and this is specially true if we consider the improvements that Termite2 presents in terms of system scalability. In fact, the scalability tests show that Termite2 is much more scalable than Termite, allowing a user to use a much larger number of emulators. Termite2 is also faster to deploy both small and large numbers of Android emulators running locally or distributed throughout multiple machines.

In short, we believe that Termite2 satisfies all the requirements that we defined in Section 1.2. Termite2 presents significant usability and scalability improvements with very acceptable performance penalties when compared with the current Termite system.

6 Conclusion

Contents

In the beginning of this work we showed that Encounter Networks provide an important shift regarding device to device communication and data sharing. Instead of relying on a central access point (router) with an Internet connection to establish communication and data transferring, mobile devices and its applications can now use short range communication technologies like Bluetooth, 802.11 WLAN or WiFi-Direct to achieve the same results when devices are near each other. However, there is lack of support tools to help in the development and test of mobile applications that follow such paradigm, i.e. encounter-based applications.

As shown in Chapter 2 the network simulation and emulation tools available today offer no support for the necessary development and testing of encounter-based applications, while tools specially designed for application testing lack the necessary network layer where tests must be performed. Thus, Termite is still the only available tool capable of providing proper support in the development and testing of encounter-based applications.

However, Termite system does not scale and it does not support a GUI. This is crucial when we consider the nature of the applications that it helps develop and test. In fact, the current version of Termite imposes a command line interface and, given that Android Studio consumes a lot of resources and Termite can run only on a single machine, users are limited on the complexity of the encounter-based applications that can be developed. To solve the above mentioned problems, we created a new version of Termite, called Termite2, with improved usability and scalability.

Termite2 presents a new distributed system architecture where the emulated network and the emulated android devices used can run on distinct machines. The user is able to create an emulated network on one machine and offload/distribute the computational load of running a large number of emulators throughout other machines. This allows the creation of much larger emulated networks where more complex applications can be developed and tested.

While Termite only offers support for a network size with a maximum of 16 emulators, our evaluation (Chapter 5) shows that Termite2 allows the creation of networks with 90 emulators (and possibly even more), while maintaining very satisfactory levels of performance. Termite2 also presents a new GUI that allows the user to see, create and model the emulated network (and all node interactions that happen within) on a real world map through interactive clicks and menu selections. This is crucial to help developers create more complex encounter-based applications while helping them better understand the network paradigm they are working with.

With Termite2's new system architecture, features and our evaluation results, Termite2 fulfills all the goals and requirements that we set for this project (Chapter 1.2) and presents itself as a proper evolution of the Termite system.

6.1 Future Work

In terms of future work, we believe that an interesting addition to the Termite2 system would be the creation of an Android Studio plug-in that would integrate the Termite2 system on the IDE itself. This would drastically improve system usability as the user would only need to interact with Android studio.

Adding support for other types of devices (besides Android emulators using WiFi-Direct) on Termite2 would also be very beneficial to extend Termite2 functionality and use cases.

Lastly another interesting addition that could be made is the implementation of automatic discoveries of Termite2 Server on a specific network, removing the need to specify the network IP address of the machines where each Termite2 Server is running.

Bibliography

[1] W3Techs, "Gsma intelligence," last accessed December 2020. [Online]. Available: https://www.gsmaintelligence.com

[2] STATISTICA, "Mobile internet usage worldwide - statistics and facts," last accessed December 2020. [Online]. Available: https://www.statista.com/topics/779/mobile-internet

[3] J. Kurhinen, V. Korhonen, M. Vapa, and M. Weber, "Modelling mobile encounter networks," *17th International Symposium on Personal, Indoor and Mobile Radio Communications*, 2016.

[4] W.-F. Alliance, "Portable wi-fi that goes with you anywhere," last accessed November 2020. [Online]. Available: https://www.wi-fi.org/discover-wi-fi/wi-fi-direct

[5] R. Bruno, N. Santos, and P. Ferreira, "Termite: Emulation testbed for encounter networks," *Mobiquitous 2015 proceddings of the 12th EAI International Conferance on Mobile and Ubiquitous System: Computing*, pp. 31–40, 2015.

[6] M. A. Khan, W. Cherif, F. Filali, and R. Hamila, "Wi-fi direct research, current status and future perspectives," *Journal of Network and Computer Applications*, pp. 245–255, 2017.

[7] GOOGLE, "Android studio the official integrated development environment (ide) for android app development," last accessed December 2020. [Online]. Available: https://developer.android.com/studio

[8] N. Santos, P. Ferreira, and R. Bruno, "Lesson 3 - simulating device movement," last accessed December 2020. [Online]. Available: https://nuno-santos.github.io/termite/index.html

[9] S. Basagni, M. Conti, S. Giordano, and I. Stojmenovic, *Mobile Ad Hoc Networking: Cutting Edge Directions*, 2nd ed. IEEE Press and Wiley, 2013.

[10] P. Koleva, P. Semov, V. Poulkov, and O. Asenov, "Layered routing algorithm for sustainable manet operation," *39th International Conference on Telecommunications and Signal Processing (TSP)*, 2016.

[11] Z. Luo, X. Gan, X. Wang, and H. Luo, "Optimal throughput–delay tradeoff in manets with supportive infrastructure using random linear coding," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 9, pp. 7543–7558, 2016.

[12] M. Conti and S. Giordano, "Mobile ad hoc networking: milestones, challenges, and new research directions," *IEEE Communications Magazine*, vol. 52, pp. 85–96, 2014.

[13] NS-2, "The network simulator - ns-2," last accessed December 2020. [Online]. Available: http://nsnam.sourceforge.net/wiki/index.php/Main_Page

[14] D. Wetherall, "Otcl - mit object tcl," last accessed December 2020. [Online]. Available: http://otcl-tclcl.sourceforge.net/otcl/

[15] NS-3, "Ns-3 network simulator," last accessed December 2020. [Online]. Available: https://www.nsnam.org

[16] L. Bajaj, M. Takai, R. Ahuja, K. Tang, R. Bagrodia, and M. Gerla, "Glomosim: A scalable network simulation environment."

[17] S. N. Technologies, "Qualnet - network simulation," last accessed December 2020. [Online]. Available: https://www.scalable-networks.com/qualnet-network-simulation

[18] OMNeT++, "Omnet++ discrete event simulator," last accessed December 2020. [Online]. Available: https://omnetpp.org

[19] J-Sim, "J-sim network simulator," last accessed December 2020. [Online]. Available: http://www.kiv.zcu.cz/j-sim/

[20] O. O. N. Performance, "Opnet network simulator," last accessed December 2020. [Online]. Available: http://opnetprojects.com/opnet-network-simulator/

[21] G. Chengetanai and G. B. O'Reilly, "Survey on simulation tools for wireless mobile ad hoc networks," *IEEE International Conference on Electrical, Computer and Communication Technologies*, 2015.

[22] S. Mallapur and S. Patil, "Survey on simulation tools for mobile ad-hoc networks," *RACST – International Journal of Computer Networks and Wireless Communications*, vol. 2, no. 2, pp. 2250–3501, 2012.

[23] EstiNet, "Estinet network simulator and emulator," last accessed December 2020. [Online]. Available: https://www.estinet.com/ns/?page_id=21140

[24] J. Quaglio, T. Gunji, and C. Hirata, "Extending nctuns simulator to support mobile networks," *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, no. 2, 2009.

[25] M. Imran, A. M. Said, and H. Hasbullah, “A survey of simulators, emulators and testbeds for wireless sensor networks,” *International Symposium on Information Technology*, 2010.

[26] A. El-Haj-Mahmoud and A. Kayssi, “Emunet: A real-time ip network emulator,” *SAC 4th Proceedings of the 2004 ACM symposium on Applied computing*, no. 4, pp. 2250–3501, 2004.

[27] C. Mark and S. Darrin, “Nist net: A linux-based network emulation tool,” *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 3, pp. 111–126, Jul. 2003.

[28] C. Enrico and D. Renzo, “The netwire emulator: A tool for teaching and understanding networks,” *SIGCSE Bull.*, vol. 33, no. 3, pp. 153–156, 2001.

[29] K. B. Dhanapal, S. Sankaran, A. A. Somasundara, and S. Paul, “Windtunnel: A tool for network impact testing of mobile applications,” *2012 Annual SRII Global Conference*, pp. 34–43, 2012.

[30] Z. Pei and N. Lionel, “Empower: a network emulator for wireline and wireless networks,” *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No.03CH37428)*, 2003.

[31] ——, “Emwin:: Emulating a mobile wireless network using a wired network,” in *Proceedings of the 5th ACM International Workshop on Wireless Mobile Multimedia*. New York, NY, USA: ACM, 2002, pp. 64–71.

[32] 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*, pp. 1–6, 2009.

[33] ——, “A multi-purpose wireless network emulator: Qomet,” *22nd International Conference on Advanced Information Networking and Applications - Workshops*, pp. 223–228, 2008.

[34] ——, “Starbed2: Large scale, realistic and real time testbed for ubiquitous networks,” *3rd International Conference on Testbeds and Research Infrastructure for the Development of Networks and Communities*, 2007.

[35] Google, “Monkeyrunner user guide,” last accessed December 2020. [Online]. Available: https://developer.android.com/studio/test/monkeyrunner

[36] J. Foundation, “Appium automation for apps,” last accessed December 2020. [Online]. Available: http://appium.io

[37] O. Source, “Expresso framwork,” last accessed December 2020. [Online]. Available: https://developer.android.com/training/testing/espresso

[38] ——, “Robotium user scenario testing for android,” last accessed December 2020. [Online]. Available: https://github.com/RobotiumTech/robotium

[39] S. Gunasekaran and V. Bargavi, “Survey on automation testing tools for mobile applications.”

[40] T. T. of Princeton University, “Planetlab an open platform for developing, deploy and accessing planetary-scale services,” last accessed December 2020. [Online]. Available: https://www.planet-lab.org

[41] T. university of Utah, “Emulab,” last accessed December 2020. [Online]. Available: https://www.emulab.net/portal/frontpage.php

[42] Google, “Android standard development kit,” last accessed December 2020. [Online]. Available: https://developer.android.com/studio

[43] ——, “Create and manage virtual devices,” last accessed December 2020. [Online]. Available: https://developer.android.com/studio/run/managing-avds

[44] ——, “Command line tools,” last accessed December 2020. [Online]. Available: https://developer.android.com/studio/command-line

[45] ——, “Set up android emulator networking,” last accessed December 2020. [Online]. Available: https://developer.android.com/studio/run/emulator-networking

[46] ——, “Create p2p connections with wi-fi direct,” last accessed December 2020. [Online]. Available: https://developer.android.com/training/connect-devices-wirelessly/wifi-direct

[47] M. Brian, “sshuttle: where transparent proxy meets vpn meets ssh,” last accessed November 2020. [Online]. Available: https://sshuttle.readthedocs.io/en/stable/index.html

[48] Google, “Google maps platform documentation,” last accessed December 2020. [Online]. Available: https://developers.google.com/maps/documentation

[49] “Termite2 tests scripts and results,” last accessed December 2020. [Online]. Available: https://github.com/nuno-santos/termite/tree/master/SourceCode/Termite2/Tests

[50] J. Dumas and B. Loring, *Moderating Usability Tests: Principles and Practices for Interacting*, ser. Interactive Technologies. Elsevier Science, 2008. [Online]. Available: https://books.google.pt/books?id=7fsRZ30GD1QC

[51] Google, “Chrome remote desktop,” last accessed December 2020. [Online]. Available: https://remotedesktop.google.com

[52] IST, “Computação móvel e ubíqua - project,” last accessed December 2020. [Online]. Available: https://fenix.tecnico.ulisboa.pt/disciplinas/CMov/2018-2019/2-semestre/project

[53] Google, “Getting started with google maps platform,” last accessed December 2020. [Online]. Available: https://developers.google.com/maps/gmp-get-started#enable-api-sdk

A

Termite2 Installation

Software Requisites

- Termite2 files/components (Termite2-Cli, Termite2Server and Termite2API); [1]
- Java SE Development Kit 8 or above; [2]
- Set the environmental variable JAVA_HOME; [3]
- Apache Tomcat Server v9; [4] [5]
- Localhost network ports available for Termite2-Cli: 8081; [5]
- Local network ports available for Termite2Server: 7000, (8085 and 8095); [6]
- Emulator instances must be provided by Android SKD;
- Most up to date Android SDK command line tools.

[1] All Termite2 files/components can be found at "https://github.com/nuno-santos/termite/tree/master/SourceCode/Termite2/Distribution"
[2] Found at "https://www.oracle.com/java/technologies/javase-downloads.html"
[3] To see how this can be done please look at "https://docs.oracle.com/cd/E19182-01/821-0917/inst_jdk_javahome_t/index.html"
[4] Found at "https://tomcat.apache.org/download-90.cgi"
[5] This is only needed if the user intends on using the Termite2 GUI.
[6] These ports can be user configured by accessing the file config/communicationports.txt on the Termite2Server folder

Introduction

Termite2 is a test-bed for developing mobile applications that use the encounter based paradigm. Termite2 system is composed by three main components:

- **Termite2-API** used on the application; it works the same way as the old Termite-API version found on the Termite system (more details in the related documentation [8]);
- **Termite2 Client** is responsible for creating and manipulating the emulated network;
- **Termite2 Server** is responsible for managing the Android emulators used on the emulated network.

Termite2 supports a distributed system architecture; this means that the emulated network provided by the Termite2 Client component can run on an independent machine while all the Android emulators can run on other machines. To achieve this the Termite2 Server must first be installed and run on the machine(s) the user intends to execute the emulators. To install each component on the desired machine(s) just follow the steps presented on the following sections.

Termite2 Server

1. Set the following environmental variables:
   - TERMITE2_SERVER_PATH="path to Termite2Server folder"
   - ANDROID_SDK_PATH="path to android studio SDK" [7]
2. Run the Termite2 Server (scripts are inside the Termite2Server folder):
   - On Linux or Mac OS - run the script termite2server.sh
   - On Windows - run the batch file termite2server.bat
3. If everything goes well the following output is expected on the terminal ,running on MacOs, on a node with local ip 192.168.1.X:

```
Termite2 Server ONLINE on network 192.168.1.X:
To register this Termite2 Server on Termite2 Client use the address: 192.168.1.X:Y
Working Directory = .../Termite2 Server
TERMITE2_PLATFORM = mac
ANDROID_SDK_PATH = .../Library/Android/sdk
Type "help" or "h" for the full command list
```

[7] Assuming a default installation, the Android SDK folder can be found at `~/Library/Android/Sdk` for MacOS, `~/Android/sdk` for Linux and `%LOCALAPPDATA%\Android\sdk` for Windows.

Termite2Server is now up and ready. Before continuing to the Termite2 Client installation take note of the address 192.168.1.X:Y presented when we first start the Termite2 Server (or perform the "help" command on the Termite2 Server console). We will use this address to register the Termite2 Server on the Termite2 Client.

Important note (Android auth_token ERRORS): For Termite2Server to be able to properly create, delete and manage emulator instances, make sure that the authentication token necessary to execute such operations is changed to null. To do this you need to access the following file ".emulator_console_auth_token". To access it on MacOs or Linux - "`nano ~/.emulator_console_auth_toke`"; On windows - "`C:\Users\USER\.emulator_console_auth_token`". After finding and opening the file delete its contents, letting the file blank. DO NOT DELETE THE FILE.

Termite2 Client

1. Set the following environmental variables:
   - TERMITE2_CLI_PATH="path to Termite2-Cli folder"
   - TOMCAT_PATH=".../apache-tomcat-9.0.34" (this variable is optional and is only needed if the user intends on using Termite2 GUI; see next section)
2. Set the Termite2 Server(s) network(s):
   - For Termite2 Client to connect with the Termite2 Server(s) we first need to indicate to Termite2 Client where each Termite2 Server is running. To do this we need to provide the address on Termite2 Server(s) used for Termite2 Client connections. This address is presented when we start the Termite2 Server or by performing the "help" command on the Termite2 Server(s) console(s).
     - Open the text file at /Termite2-Cli/config/networks.txt

       Here you must type each Termite2 Server address that you wish the Termite2 Client to connect to, example:

       192.168.1.1:8085

       192.168.1.2:8085

       192.168.1.3:8085

       ...
3. Run the Termite2 Client (scripts are inside the Termite2-Cli folder):
   - On Linux or Mac OS - run the script termite2cli.sh
   - On Windows - run the batch file termite2cli.bat

4. If everything goes well the following output on the terminal is expected:

```
Connection/s to Termite2Server/s established without errors.
Termite2 UI Options:
Type '1' to use the default console UI.
Type '2' to use the Web-page GUI.
```

Termite2 Client is now up and ready; the user can now choose the desired user interface. Option 1 for the default console UI or option 2 to use the Termite2's graphical user interface (Termite2 GUI) that runs on Tomcat and is accessed through the URL: localhost:8080/Termite2UI/index.html; to set up this new user interface please see the next section.

Termite2 GUI [8]

The new Termite2 user interface uses the Google maps API in order to better display the virtual nodes created on the emulated network and their interactions. Because of this it is necessary to first set the required Google Maps API key with a set of active APIs. This is done as follows:

1. To get the API key, and **activate the necessary APIs, Roads API, Directions API and Maps JavaScript API**, please follow the official google documentation [53].

2. **After generating your key and activating the APIs** place your API key (ONLY THE KEY) inside the text file at ".../Termite2-Cli/ui/Termite2UI/apikey.txt".

Everything is set and you can now use the new Termite2 GUI by selecting option 2 when starting the Termite2 Client.

[8]This section is optional and it is only needed if the user intends to use Termite2 GUI.

B

Termite2 Client - Console Commands

This document presents and explains the available Termite2 commands that can be performed on the Termite2 console interface. The available commands are presented in 3 groups. Network commands are those used to create and model the emulated network (these commands also exist on the previous system, Termite). Emulator commands are used to interact and manage the emulators used on the emulated network. Other commands can be seen as miscellaneous commands with varied uses. For each group, the commands are identified by their names, followed by their syntax and a brief description/explanation. See example bellow:

- **cmd**
  - Syntax (cmd <arg1> <arg2> <arg3> ...).
  - Description and explanation of command cmd.

Network commands

- **newdevice**
  - newdevice <node>
  - Creates a new virtual device/node on the emulated network with <node> as its name/id.
- **deletedevice**
  - deletedevice <node>
  - Deletes the <node> from the emulated network.
- **binddevice**
  - binddevice <node> <emu_id>
  - Binds the <node> to the emulator <emu_id>. [1]
- **unbinddevice**
  - unbinddevice <node> <emu_id>
  - Unbinds the <node> to the emulator <emu_id>.
- **move**
  - move <node1> (<node2>, <node3>, . . .)
  - Emulates the movement of <node1> to the vicinity of the nodes <node2> and <node3>. This makes the nodes <node2> and <node3> neighbours of <node1>. Note that the command does not make <node2> and <node3> neighbours of each other.
    To emulate a node moving out of the vicinity of all other do "move <node1> ( )".
- **creategroup**
  - creategroup <go_node> (<node2>, <node3>, . . .)
  - Emulates the creation on a peer-to-peer group where the <go_node> (group owner node) creates a group with the nodes <node2> and <node3> as members (himself included). For the group creation to work the member nodes (<node2> and <node3>) must be in the vicinity of the <go_node>. Note that is possible to create empty groups by not providing members, example: "creategroup <go_node> ( )".

[1] <emu_id> is the id given by the Termite2 Client to the emulators. To see the emulators, their id's and other data use the command "list emus". This command is explained on Other commands group.

- **joingroup**
  - joingroup <node> (<go_node1>, <go_node2>, . . .)
  - Emulates <node> joining the group(s) <go_node1> and <go_node2> (. . .). Note that <node> must be in the vicinity of the go nodes <go_node1> and <go_node2>.
- **leavegroup**
  - leavegroup <node> (<go_node1>, <go_node2>, . . .)
  - Emulates <node> leaving the group(s) <go_node1> and <go_node2> (. . .).
- **deletegroup**
  - deletegroup <go_node>
  - Deletes the group corresponding to the <go_node>.
- **ping**
  - ping
  - Pings all emulators in order to check if WiFi-Direct is on (through the Termite2 API on the applications).
- **commit**
  - commit
  - Commits the emulated network state/topology on the Termite2 Client to all emulators that are bound to nodes.

Emulators commands

- **refresh**
  - refresh
  - Refreshes the available emulator detected by the Termite2 Client.
- **installed**
  - installed
  - Returns all the AVD's (Android virtual devices) installed on the connected Termite2 Server.

- **createavds**
  - createavds <t2server-ip> <nº> <avd_name> <api>
  - This command creates <nº> AVD's with the name <avd_name> and api number <api> on the Termite2 Server on the address <t2server-ip> [2]. An example of this command can be "createavds 192.168.1.2 3 test 21", this creates 3 AVD's, test1, test2 and test3 all with the Android api 21 on the Termite2 Server at 192.168.1.2.
- **destroyavds**
  - destroyavds <t2server-ip> <avd_name1> <avd_name2 > . . . / all
  - This command deletes the AVD's <avd_name1> and <avd_name2 > (. . .) on the Termite2 Server on the address <t2server-ip>. If you want to delete all AVD's on a Termite2 Server machine you can use the argument "all" instead of passing the AVD names (for example, "destroyavds 192.168.1.2 test1 test2" or "destroyavds 192.168.1.2 all").
- **startemus**
  - startemus <t2server1-ip> <all/nº> <app-pack> | ...
  - This command starts <nº> emulators (or all available if we use "all", available means installed AVDs) on the Termite2 Server on the address <t2server1-ip> and optionally, it automatically starts the application with the package name <app-pack> on the emulators after they start. We can chain multiple startemus commands for multiple Termite2 Servers by using the operator "|", for example: " startemus 192.168.1.2 2 test.package | 192.168.1.3 all " starts 2 emulators running the application with package test.package on the Termite2 Server - 192.168.1.2 and at the same time start all available emulators (with no app) on the Termite2 Server - 192.168.1.3. Note if we try to start a number (<nº>) of emulators that exceeds the number of installed AVDs on the Termite2 Server, the command resolves to starting the maximum number of emulators possible. For example if we try to start 5 emulators but there is only 3 AVDs installed, the command starts all 3.
- **stopemu**
  - stopemu <emu_id>
  - Stops chosen emulator <emu_id>.

[2]<t2server_ip> is the address that identifies the termite2 Server targeted by the command. To see the address of all the Termite2 Severs use the command "list servers". This command is explained on Other commands group.

- **stopall**
  - stopall / stopall <t2server1-ip> <t2server2-ip> ...
  - This command stops all emulators on the Termite2 Server at <t2server1-ip> and <t2server2-ip> (...). If the command is performed without arguments we stop all emulators on all connected Termite2 Servers.
- **installapp**
  - installapp <t2server-ip> <apk-name> <avd_name1> <avd_name2> / all | ...
  - This command installs the apk <apk-name> [3] on the online emulators [4] <avd_name1> <avd_name2> (...) running on the Termite2 Server at <t2server-ip>. If we want to install the apk on all the online emulators on the Termite2 Server at <t2server-ip> we can simply use the "all" argument instead of the AVD names. Similar to the startemus command, we can use the operator "|" to chain multiple installapp commands on multiple Termite2 Servers, for example: "installapp 192.168.1.2 test.apk test1 test2 | 192.168.1.3 test.apk all" install the apk "test.apk" on the emulators test1 and test2 at the Termite2 Server - 192.168.1.2 and at the same time it installs the same apk on all online emulators at the Termite2 Server - 192.168.1.3.
- **startapp**
  - startapp <t2server-ip> <app-pack> <avd_name1> <avd_name2> / all | ...
  - This command starts the app with the package name <app-pack> on the online emulators <avd_name1> <avd_name2> (...) running on the Termite2 Server at <t2server-ip>. If we want to start the apk on all the online emulators on the Termite2 Server at <t2server-ip> we can simply use the "all" argument instead of the AVD names. Similar to the startemus and installapp commands, we can use the operator "|" to chain multiple startapp commands on multiple Termite2 Servers, for example: "startapp 192.168.1.2 test.package test1 test2 | 192.168.1.3 test.package all" starts the application corresponding to the app package "test.package" on the emulators test1 and test2 at the Termite2 Server with the address 192.168.1.2 and at the same time starts the same app on all online emulators at the Termite2 Server with the address 192.168.1.3.

[3]This file of type .apk must be stored on the folder ~/Termite2Server/apks on the respective Termite2 Server(s).
[4]Online emulators are emulators that are already running on the Termite2 Server machine(s)

Other commands

- **list**
  - list emulators / servers / devices / groups / neighbours / binds / tnetwork / scripts / history
  - This command lists various types of information based on the argument; “emulators” shows the emulators currently detected by the Termite2 Client; “servers” shows the IP addresses of the connected Termite2 Server(s); “devices” presents the virtual nodes on the emulated network; “groups” shows the peer-to-peer groups that exist on the emulated network; “neighbours” list the neighbours of each virtual node on the network; “binds” shows the binds that currently exist between nodes and emulators; “tnetwork” shows the state of each virtual node on the network (their neighbours and groups); “scripts” presents the name of all the scripts files available to be loaded with the load command; “history” shows the command history.

- **load**
  - load <scrip_name.txt>
  - This command loads an txt file [5] that contains Termite2 commands (one per line) and executes them in order. This is useful to easily execute a large number of commands or save/load various operations. For example, we can create a script that starts multiple emulators on various Termite2 Servers. Then an application is installed and started on all emulators. In this example, this allows us to easily set up emulators ready to be used on the emulated network. While the file is being loaded (commands are being processed) you can type “stop” on the Termite2 Client command line to stop the load command.

- **wait**
  - wait <time> ms/s/m
  - This command stalls the program for a period of <time>. This time can be in milliseconds “ms” (default, if no argument is passed), seconds “s” or minutes “m”. This command is more useful if used on a script that runs multiple commands on the Termite2 Client (using the command load) and applies waiting times between some commands. For example imagine a txt file with the following commands, on per line: newdevice a, newdevice b, move a (b), wait 5 s, move a ( ). Loading this file (load command) will cause the creation of the nodes a and b, then emulates node a coming to the vicinity of node b, waiting 5 seconds and the node a moves away from the vicinity of node b.

[5] The file has to be of type .txt and must be stored on the folder ∼/Termite2-Cli/scripts

- **localscript**
  - localscript <file_name> <arg1> <arg2> ...
  - This command allows us to execute the script file <file_name> with the arguments (for the script) <arg1> and <arg2> (...) from the Termite2 Client console. This can be used to execute scp scripts that transfers files to the Termite2 Server(s) running remotely (for example, sending apk files to be be stored on the Termite2 Server that can then be installed on emulators).
- **runscript**
  - runscript <t2server-ip> <file_name> <arg1> <arg2> ...
  - This command is similar to the localscript command, but the script is run on the Termite2 Server at <t2server-ip>. The script file must be of type .sh or .bat and must be stored on the folder ~/Termite2Server/scripts on the respective Termite2 Server. This command is useful if you want to run scripts on Termite2 Server machines remotely from a single control point (the Termite2 Client).
- **time**
  - time set / log / show / reset
  - This command works like a stop watch. Use "set" to initiate the time and then use "log" to log and see how much time as passed in milliseconds. Use "show" to display all logged values since the last "set". Use "reset" to reset the time watch and the "show" logged values.
- **cls**
  - cls
  - Clear the console screen.
- **help**
  - help commands/cmds
  - Presents a list with all available commands. If you pass the argument "commands" or "cmds" the commands are listed with a brief explanation.
- **quit**
  - quit
  - Terminates the Termite2 Client.