Termite2 - Supporting Scalable and Usable Encounter-based Apps

Fernando Daniel Alves Moreira
Instituto Superior Técnico
Lisboa, Portugal
fernando.moreira@tecnico.ulisboa.pt

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 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 an 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 [1] can run on Windows, Linux or Mac, it supports Android mobile applications and virtual devices.

Keywords: Termite2, Encounter Networks, Android Development, Wi-Fi Direct, Virtual devices.

1 INTRODUCTION

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 [13].

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 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. Unfortunately, there is a lack of tools that support the proper development and test of applications based on the encounter networks paradigm.

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 they are near each other. Assuming that Alice wants to support encounter networks, she may choose an already available communication technology for this paradigm, e.g., Wi-Fi Direct, as the communication technology for her application.

When developing her app, scalability is one of Alice application requirements. Her application should be equally capable of handling both small and large number of users connected at the same time. With the currently available solutions, Alice would have to gather dozens of Android devices in order to accurately test her 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.

In fact, there is a lack of solutions capable of combining the emulation/simulation of encounter networks (with proper multi-node emulation and displacement) and on top of this network utilize emulated mobile devices (for our example Android devices) to create testing scenarios for encounter based applications.

Our related work (Section 2) shows that network simulation and emulation tools available today, while able to offer the necessary

[1]Termite2 source code, distribution files and documentation are publicly available at https://github.com/nuno-santos/termite/tree/master/SourceCode/Termite2 [15]

network layer capable of simulating/emulating encounter networks, offer no support neither for virtual devices nor for testing encounter-based applications on top of the network created. At the same time, tools specially designed for application testing similar devices lack the necessary network layer where the tests can be performed.

The only tool available today capable of properly supporting the development and test of encounter based applications is the Termite system [3] [20]. 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 discuss on the related work section of this document, Termite shows a level of system usability and scalability that is not satisfactory when we consider the nature of the applications that it helps to develop and test. This motivates us to create Termite2, an evolution of the previous Termite system, with improved system scalability and usability. Termite2 presents a new distributed architecture that allows a user to create and model the emulated encounter on one machine and distribute the computational load of running a large number of emulators across multiple different machines. Termite2 also presents a new graphical user interface (GUI) where the emulated encounter network is created, modeled and presented on a real world map. Managing the emulators used on the emulated network can also be done from this interface through interactive button prompts and selections.

Thus, this work provides the following contributions:

- Architectural design and creation of Termite2 test bed (the next version of the Termite system with new system interactions and protocols);
- Creation of the Termite2 Server components which allows us to manage and use Android emulators running across multiple local or remote machines;
- Design and creation of Termite2's new graphical user interface;
- Evaluation of Termite2 that presents major improvements in terms of system usability and scalability, when compared to Termite.

The rest of this document is organized as follows. Section 2 describes and evaluates different systems related to Termite2 goals. Section 3 presents our project solution, describing Termite2 background (Termite) and its architecture. Section 4 presents the implementation of Termite2's new components (Termite2 Server and Termite2 GUI). Section 5 presents the evaluation of the system. Finally, Section 6 presents some conclusions.

2 RELATED WORK

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. Therefore, for our related work we focus on the tools/solutions that target each one of these layers, thus addressing Network Simulators, Network Emulators, and Test Frameworks.

2.1 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).

NS-2 [16] is an open source, object oriented TCL (OTcl [23]) script interpreter with a network simulation event scheduler. This network simulator is used for setting up and running wired or wireless network simulations that can be used to perform various network tests. In order to do this, a user can write a simulation program usnig the 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 encounter network. NS-2 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 a user to see the network and all node interactions. Sadly the 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. However, the real problem that renders this tool incapable of supporting our project is the fact that NS-2 has no support for mobile application development and testing, since it does not support mobile virtual devices as network nodes of the simulated network.

NS-3 [17] 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 top of the simulated network, thus making this tool not suitable for our project needs.

We could continue discussing other Network Simulation tools, e.g., GloMoSim [1], OMNET++ [18], J-Sim [12], or OPNET [19]. Each one of these tools provide distinct advantages and disadvantages [4] [14]. Despite any desirable quality all these systems may have to this project, they all fail in providing support for mobile application development and testing on top of the simulated network; this is crucial for this project given its fundamental objective of helping in the development of encounter-based applications.

The reason for disadvantages this comes from the concept of network simulation itself. Network simulation tools are not concerned with what is being developed and tested on top of the simulated network. Instead, these tools focus on simulating and testing the characteristics of the network itself in order to support its design and development.

2.2 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; thus, such computers can act exactly as if they were attached to a real network [11]. This allows a user to accurately gauge an application's responsiveness, throughput, and quality of 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. Most network emulation tools also provide the necessary network characteristics to emulate encounter networks by allowing the network nodes to move within the network. Unfortunately, all the network emulation tools that we analysed for this project show no support for emulated mobile devices. Nevertheless, some of the tools present interesting solutions to system scalability; below we present one of them.

NetWire [5] 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, which 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 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 where real computation nodes. Network and node emulation can be spread among multiple workstations to distribute the computational load, by connecting several network servers, drastically improving the system scalability.

Unfortunately, when evaluating this system for our project needs, NetWire has no GUI, it 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 Section 3 we see that our project architecture is inspired by NetWire client-server solution, and will use the same 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.

2.3 Test Frameworks

In the previous sections we have focused on presenting tools that target the network layer as a basis for the development of encounter based applications. In this section we discuss tools that target the development and testing of mobile application and see if they support the necessary network layer to simulate peer-to-per tests.

MonkeyRunner [9] 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. In fact, 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 the encounter network paradigm. However, the complexity and work needed to integrate MonkeyRunner and the proper network emulation tool would require a huge engineering effort.

There exists a lot of commercial and open source solutions to provide automated testing of mobile application. We have tools like Appium [6], Expresso [21] and Robotium [22]. All of these tools present distinct advantages and disadvantages [10], and we could discuss each one of them in detail. It is, however, unnecessary because all these tools show the same problem as the one found in MonkeyRunner. In fact, none is capable of emulating/simulating a network, which in turn renders them unable to properly execute tests for the particular case of encounter based applications.

2.4 Termite

Termite [3] [20] is an emulation test-bed created to provide support for the development and testing of mobile applications that apply the encounter networks paradigm. Termite is the only system that has the ability to properly test mobile applications running on emulated mobile devices on top of an emulated encounter network, with proper support for node displacement and interactions. Thus, Termite presents itself as an obvious starting point for this work. However, the system is not a perfect solution and upon usage, it clearly shows low levels of usability and scalability which do not satisfy our project requirements.

Termite system scalability is poor for three main reasons: i) Termite's reliance on Android studio or user made scripts to create and manage all emulator instances; ii) Android SDK [**?**] (per default configuration) can only manage a maximum of 16 emulator instances running at the same time; and iii) emulating a large encounter network requires a large number of emulators, which requires a considerable amount of computational power to run (CPU and RAM).

Thus, 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 the Termite, Android Studio and all the necessary Android emulators, if not powerful enough, will show a significant slowdown (or some emulator crashes). Therefore, we think that the current version of Termite is only suitable for testing mobile applications on small networks; this can be seen as a major system flaw when we consider the nature of the applications that Termite is helping to develop.

In terms of system usability we think that Termite presents two major flaws: i) the system does not provide a proper GUI, and ii) it does not allow the user to directly manage the virtual Android devices without the usage of Android studio or a 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

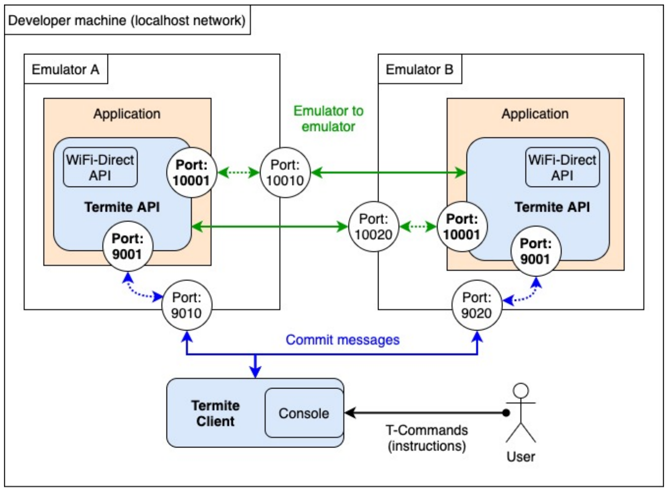


Figure 1: Termite's components and their interactions.

of emulators to virtual nodes, etc. This lack of a proper graphical user interface to help to 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 serious negative impact on usability, specially when we considering a large emulated network with a large number of emulators.

3 SOLUTION

As mentioned before on Section 1, the current Termite system is the basis for our work. Thus, it is important to first briefly describe Termite's system architecture and its components, and then present the new Termite2 main aspects.

3.1 Termite Background

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: Termite Client and Termite API. Termite Client is Termite's main component; it starts from a console window on the developer machine and supports all user interactions with the system 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) that implements most of the WiFi-Direct API [7] and must be used by the developer on the applications that they wish to develop/test using Termite. These applications run on emulators on the developer machine.

On Termite (see Fig. 1), Android emulators are identified with two addresses: localhost and a port number. For example, in Fig. 1 the target emulator A is identified by the addresses `localhost:9010` and `localhost:10010`. These addresses are set by the Termite Client with the help of a configuration file configured by the developer (on this file the user must specify the ports that she/he wishes to use for the emulators). These addresses then enable two types of interactions between Termite's system components: Termite Client to emulators, and emulator to emulator communication. Note that with Termite, all these interactions happen locally on the developer machine (localhost).

The Termite Client communicates with the target emulators A and B, by sending messages respectively to `localhost:9010` and `localhost:9020`, which are then redirected[2] to the application (running inside the emulators) and received by a socket server opened by the Termite API on port 9001 (within the enclosed network environment of each emulator). Messages are sent from Termite Client to the emulators following instructions given by the user on the Termite Client. These messages are called Commit messages and 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).

Commit messages contain information that allows the Termite API to trigger peer-to-peer events inside an application using the WiFi-Direct API provided by Android [7]. One of such events is the creation of a peer-to-peer group with other target emulators. When this happens, applications can use the Termite API library to create a socket connection with other group members, allowing them to communicate (green arrows in Fig. 1). The target emulator A connects with the Target emulator B using the address `localhost:10020`. This connection is then redirected [2] to the application running inside target emulator B and received by a socket server also opened by the Termite API on port 10010.

3.2 Termite2

Termite2 architectural solution solves Termite's usability and scalability limitations. Thus, Termite2 presents two major system differences when compared to 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) accessed via a web browser that displays a visual representation of the created emulated network on a real world map; this interface allows a user to interact with Termite2 via clicks and menu choices.

The above mentioned differences are expressed on Termite2 architecture through the creation of two new system components:

[2] These redirections are set by the Termite Client on the emulators using the available Android SDK command line tools and enable an outside application, like the Termite Client, to send messages to applications running inside an Android emulator.

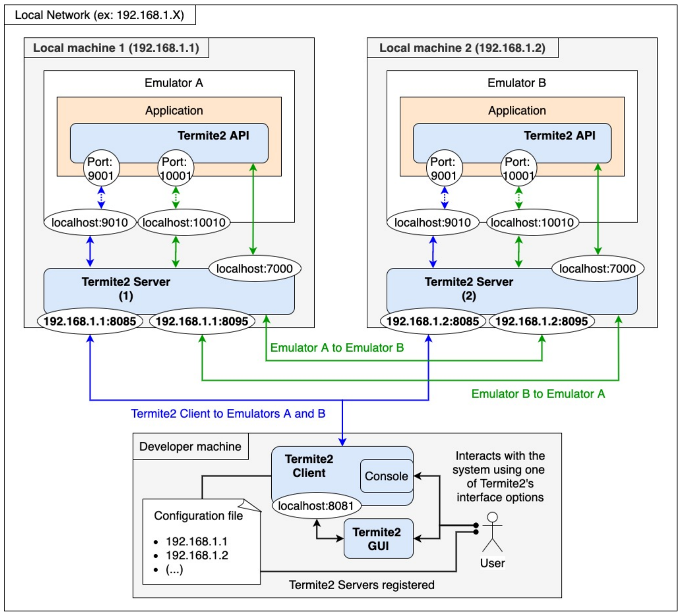


Figure 2: Termite2's components and their interactions.

Termite2 Server, and Termite2 GUI. These new components work alongside the old Termite's components, Termite Client and Termite API; the corresponding components in Termite2 are called Termite2 Client and Termite2 API, respectively. Therefore, Termite2 is comprised by the interaction of the following components: Termite2 Client, Termite2 Server, Termite2 API, and Termite2 GUI. Fig. 2 shows Termite2's components and their interaction.

Termite2 Client runs from a console window on the developer machine where the emulated network is created and modelled. User interactions with Termite2 are also performed through this Client; however, the user is now able to choose between two interface options: the old console interface where system interactions are done using written 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 (on Termite system).

Just like with Termite, on Termite2 there are still two types of communication to consider: from Termite2 Client to emulators, and from emulator to emulator. However, these communications are no longer performed in the same way as they where on Termite. Fig. 2 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, e.g., using sshuttle [2] [3] 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. 2).

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

3.2.1 *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 the Termite Client did. However, an important change is that emulators are now also identified by the machine IP address where they are running. For example, in Fig. 2, 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. 2, 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. 2).

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 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. 2 we can see this and the addresses used by looking at the green arrows.

Lastly, it is important to present another capability of the Termite2 Server; it 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 a user to create and manage a large number of emulators (and the applications running inside) distributed across multiple machines from a single control point.

3.2.2 *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. It can be accessed via a web browser and in order for the Termite2 GUI to exchange data with the Termite2 Client, it connects to it through a socket server that runs on the Termite2 Client on `localhost:8081` (we show this connection on Fig. 2).

Visually, this new interface is divided in three distinct parts (see Fig. 3). The first one is the Network view, where the user is able to create and model the emulated network on a real world map

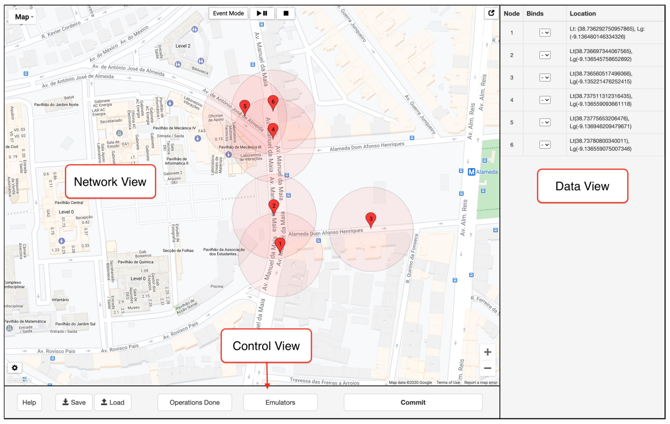
[3]sshuttle allows us to forward the IP addresses and ports of an entire remote network to our local network

(provided by Google Maps) through interactive mouse clicks, drags and menu choices. The virtual network nodes are also presented on this view (red markers in Fig. 3). 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 a user to perform a number of tasks on the interface.

With Termite2 GUI we are able to emulate node movement on the Network view 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.

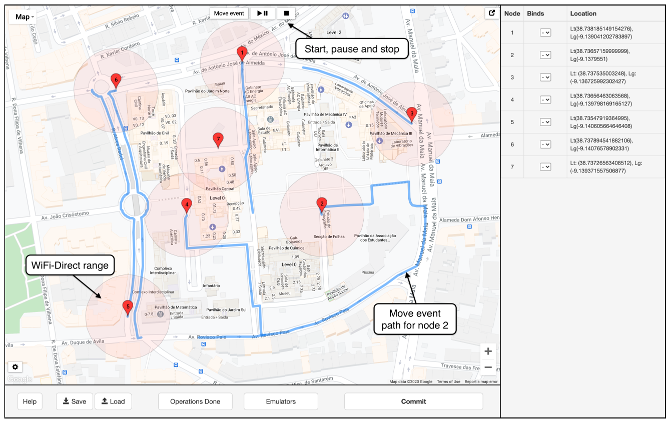
Move events allow users to create automatic movement paths for each network node with a fixed movement speed that emulates walking, cycling or driving (see Fig. 4). When nodes move via these events they automatically create/join peer-to-peer groups when in range of other nodes (node range is displayed 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 a 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 scenario on the 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



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: Termite2 GUI views.



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 4: Termite2 GUI move events.

via a socket connection to Termite2 Client on `localhost:8081`. Messages are received by the Termite2 Client and sent to the target emulators through the Termite2 Server(s).

4 IMPLEMENTATION

In this Section, 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.

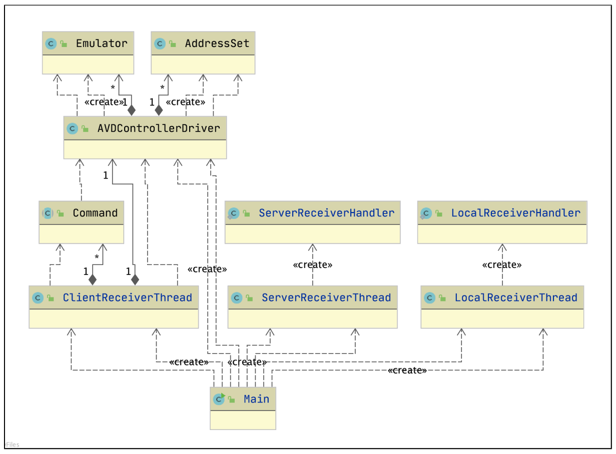


Figure 5: Simplified UML diagram of Termite2 Server Java classes.

Termite2 Server was implemented using Java. We use this language to guarantee that this component can easily run on Windows, Mac or Linux and to be consistent with the rest of Termite code base (which is already in Java). Fig. 5 shows a UML diagram of the Termite2 Server Java classes (for simplicity we only show the more relevant classes and no field 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 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 in Fig. 2 on Section 3). 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 (as shown in Fig. 2 on Section 3 (green arrows)).

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 [8]. 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. Most of the code used translates the already existing command logic on the Termite console to interactive button clicks and menu choices. Thus, we will only present the implementation aspects of the new automatic node movement option that the Termite2 GUI provides (presented in the previous Section 3.2.2).

The new movement option, 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 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). 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. 6 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).

```
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 6: Interpolation function.

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 note that 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.

5 EVALUATION

In this section we present Termite2 evaluation. The tests evaluate Termite2's main requirements: usability 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.

5.1 Usability Tests

To compare Termite2's system usability against Termite's is to compare both system interfaces, as is it through here that all interactions with the system are made. Therefore, to evaluate and compare the usability of both Termite and Termite2 we developed a tests guide [15]. 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. 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 correspond 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 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 [15] 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 believe that the results obtained still allow us to take valuable conclusions over the usability of both systems and their interfaces.

Our usability tests results [15] 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 clearly presents an evolution over Termite console drastically improving system usability.

5.2 Scalability Tests

For our project, 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.

When using Termite, we need to use 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 size of 16 target emulators. This number assumes that the local machine has all the resources needed for that purpose.

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 limitations on the maximum number of emulators we can run locally. This is done by allowing the user to 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 two tests (Local Deployment and Distributed Deployment) to see how long it takes to launch an increasing numbers of emulators on two different environments.

5.2.1 Local Deployment. This test consisted in deploying an increasing number of emulators (from one to ten emulators[4] on a single machine and measure the time of deployment. We performed these tests on Termite and Termite2 with all components of both systems running on a single RNL computer.[5] 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 80 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.

The results obtained for this test are shown in Fig.7 (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.

[4]We chose ten emulators as the maximum number due to memory constrains on the testing machines.
[5]RNL machines are computers available at the IST laboratory 13.

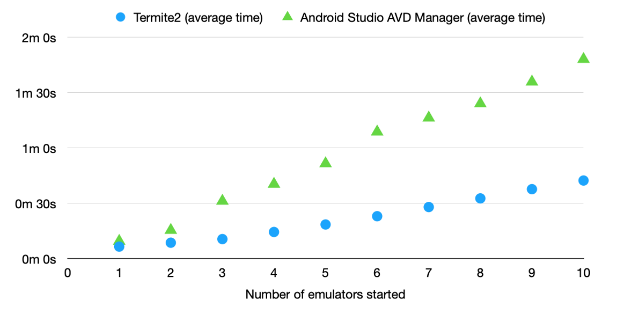


Figure 7: Local Deployment results for Termite and Termite2

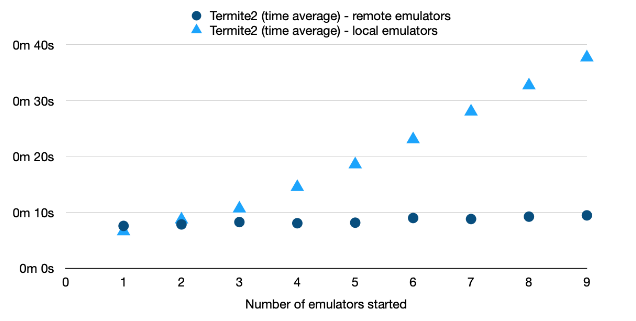


Figure 8: Startup time for local and remote emulators using Termite2

5.2.2 Distributed Deployment. To truly show how Termite2 improves system scalability we developed a test similar to the previous one but now each emulator instance is deployed on a unique RNL machine. With only 9 RNL machines available, we were able to reach a total number of 9 emulators (with one emulator running per machine), which then allows us to compare these results to those obtained on the Local Deployment test (with Termite2 on a single machine).

We perform the same test but this time we deployed 10 emulators instances per RNL machine. 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.8 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.

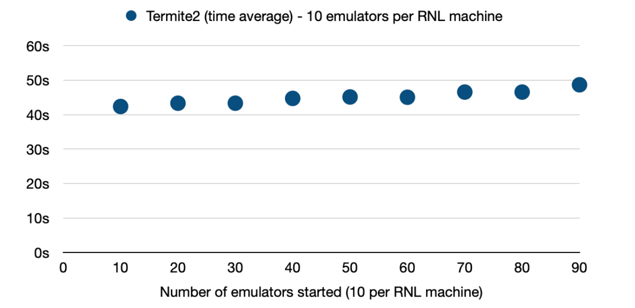


Figure 9: Maximum number of emulators we are able to use with 9 RNL machines

This parallel processing is again shown on the results obtained when starting ten emulators across the nine RNL machines (see Fig.9). 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).

By looking at Fig.9 and comparing the results obtained to those presented on the Local Deployment test we can also see that starting 90 emulators with Termite2 across 9 machines takes less time than to start 10 emulators on Termite on a single machine (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.

6 CONCLUSION

On one hand, network simulation and emulation tools available today offer no support to the necessary development and testing of encounter-based applications. On the other hand, tools specially designed for such applications development and testing lack the necessary network layer where tests must be performed. Termite is still the only available tool capable of providing proper support in the development and testing of encounter-based applications. However, Termite does not scale and it does not support a GUI.

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 the 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 (Section 5) shows that Termite2 allows the creation of networks with 90 emulators (and possibly even more). 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 and features, as shown in our evaluation, Termite2 fulfills our project goals and the system presents itself as a proper evolution of the Termite system.

REFERENCES

[1] Lokesh Bajaj, Mineo Takai, Rajat Ahuja, Ken Tang, Rajive Bagrodia, and Mario Gerla. [n. d.]. GloMoSim: A Scalable Network Simulation Environment. ([n. d.]).

[2] May Brian. [n. d.]. sshuttle: where transparent proxy meets VPN meets ssh. https://sshuttle.readthedocs.io/en/stable/index.html Last accessed November 2020.

[3] Rodrigo Bruno, Nuno Santos, and Paulo Ferreira. 2015. Termite: Emulation Testbed for Encounter Networks. *Mobiquitous 2015 proceddings of the 12th EAI International Conferance on Mobile and Ubiquitous System: Computing* (2015), 31–40.

[4] Gibson Chengetanai and Grant Blaise O'Reilly. 2015. Survey on simulation tools for wireless mobile Ad Hoc Networks. *IEEE International Conference on Electrical, Computer and Communication Technologies* (2015).

[5] Carniani. Enrico and Davoli. Renzo. 2001. The NetWire Emulator: A Tool for Teaching and Understanding Networks. *SIGCSE Bull.* 33, 3 (2001), 153–156.

[6] JS Foundation. [n. d.]. Appium automation for apps. http://appium.io Last accessed December 2020.

[7] Google. [n. d.]. Create P2P connections with Wi-Fi Direct. https://developer.android.com/training/connect-devices-wirelessly/wifi-direct Last accessed December 2020.

[8] Google. [n. d.]. Google Maps Platform Documentation. https://developers.google.com/maps/documentation Last accessed December 2020.

[9] Google. [n. d.]. Monkeyrunner User Guide. https://developer.android.com/studio/test/monkeyrunner Last accessed December 2020.

[10] S Gunasekaran and V Bargavi. [n. d.]. Survey on automation testing tools for mobile applications. ([n. d.]).

[11] Muhammad Imran, Abas Md Said, and Halabi Hasbullah. 2010. A Survey of Simulators, Emulators and Testbeds for Wireless Sensor Networks. *International Symposium on Information Technology* (2010).

[12] J-Sim. [n. d.]. J-Sim Network Simulator. http://www.kiv.zcu.cz/j-sim/ Last accessed December 2020.

[13] Jani Kurhinen, Vesa Korhonen, Mikko Vapa, and Matthieu Weber. 2016. Modelling Mobile Encounter Networks. *17th International Symposium on Personal, Indoor and Mobile Radio Communications* (2016).

[14] Sujata Mallapur and Siddarama Patil. 2012. Survey on Simulation Tools for Mobile Ad-Hoc Networks. *RACST – International Journal of Computer Networks and Wireless Communications* 2, 2 (2012), 2250–3501.

[15] Fernando Moreira, Rodrigo Bruno, Nuno Santos, and Paulo Ferreira. [n. d.]. Termite2 Source Code. https://github.com/nuno-santos/termite/tree/master/SourceCode/Termite2 Last accessed December 2020.

[16] NS-2. [n. d.]. The Network Simulator - ns-2. http://nsnam.sourceforge.net/wiki/index.php/Main_Page Last accessed December 2020.

[17] NS-3. [n. d.]. NS-3 Network Simulator. https://www.nsnam.org Last accessed December 2020.

[18] OMNeT++. [n. d.]. OMNeT++ Discrete Event Simulator. https://omnetpp.org Last accessed December 2020.

[19] OPNET Optimum Network Performance. [n. d.]. OPNET NETWORK SIMULATOR. http://opnetprojects.com/opnet-network-simulator/ Last accessed December 2020.

[20] Nuno Santos, Paulo Ferreira, and Rodrigo Bruno. [n. d.]. Lesson 3 - Simulating device movement. https://nuno-santos.github.io/termite/index.html Last accessed December 2020.

[21] Open Source. [n. d.]. Expresso framwork. https://developer.android.com/training/testing/espresso Last accessed December 2020.

[22] Open Source. [n. d.]. Robotium User scenario testing for Android. https://github.com/RobotiumTech/robotium Last accessed December 2020.

[23] David Wetherall. [n. d.]. Otcl - MIT Object Tcl. http://otcl-tclcl.sourceforge.net/otcl/ Last accessed December 2020.