EdgeEmu - Emulator for Android Edge Devices [⋆]

Lyla Naghipour Vijouyeh[1][0000−0003−2377−5864],
Rodrigo Bruno[2][0000−0003−1578−5149], and
Paulo Ferreira[1][0000−0003−0942−6289]

[1] University of Oslo, Norway, `[lylan, paulofe]@ifi.uio.no`
[2] INESC-ID/Técnico, ULisboa, Portugal, `rodrigo.bruno@tecnico.ulisboa.pt`

Abstract. The number of mobile devices is rapidly outgrowing the current world population, making them the most popular medium to communicate and share information. In addition, applications that enable communication and data sharing still heavily rely on centralized networks. We believe that this problem is mainly due to the lack of tools to help programmers develop and test applications with many devices in edge environments.
To help programmers develop and test such distributed applications, we propose EdgeEmu, an Android distributed emulation testbed for mobile applications. EdgeEmu supports a high number of Android emulators participating in a large network by allowing them to remotely participate in the emulation, thus removing the scalability bottleneck that current Android testing infrastructure has. EdgeEmu is, therefore, not limited to locally deployed emulators as opposed to the standard Android SDK.
To study the performance of EdgeEmu, extensive evaluation through different scenarios has been conducted. Results demonstrate that EdgeEmu outperforms the standard Android SDK by approximately 59.1% in terms of emulation startup time when ten Android emulators are used. Evaluations also show promising results for low latency and negligible overhead when sending messages to and from different emulators.

Keywords: Android · Edge Networks · Wi-Fi Direct · Emulation · Peer-to-Peer · Bluetooth · Edge.

1 Introduction

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

To achieve their full potential, mobile devices require communication with other mobile devices or external resources. As a consequence, almost every mobile application uses some form of communication technology. This is particularly true for data-sharing applications and location-based multiplayer games [3, 4]. These

⋆ Supported by Department of Informatics, University of Oslo. Artifacts available in https://zenodo.org/record/XXXXXXX.

applications normally provide data sharing functionalities and users communications through the Internet, using Wi-Fi or broadband cellular network. For example, suppose two users want to share a 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. Using remote services instead of local ones increases latency, and globally increases the utilized network bandwidth. In addition, such applications have the limitation that they do not function properly in case of intermittent or limited Internet connection.

To avoid using an Internet connection when users are in proximity of each other, edge networks[1] 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 use Peer-to-Peer (P2P) communication technologies like Bluetooth or Wi-Fi Direct [5] to achieve the same results when devices are near each other. Using such networks, applications can be developed to exploit user proximity.

When developing applications that take advantage of such edge networks, developing an application becomes an issue. One option is to gather dozens of Android devices in order to accurately develop it, while also having some device displacement to simulate users moving in and out of range with each other. This solution is not practical for several reasons such as cost, time, logistics, etc.

Android provides its own development tool kit with support for virtual emulation of Android devices called Android Studio [6]. However, this tool does not provide the necessary support to develop and test applications that apply the edge network paradigm. The Android Standard Development Kit (SDK) [7] does not implement multi-node emulation and displacement for the emulated Android devices, which is required to properly test edge network scenarios. Also, Android SDK supports up to 16 emulated Android devices, which limits the size of the emulated network. Other available network simulation and emulation tools (discussed in Section 2), offer no support for developing and testing such applications on top of the network created.

This lack of proper support for developing and testing edge-based applications forces developers to publish applications without proper testing or instead, drives developers away from using P2P communication technologies in favor of centralized communication technologies. To empower developers with the ability to test P2P communication-enabled applications with many Android devices, we propose EdgeEmu, a system capable of creating both small and large emulated edge networks, and support the development and testing of Android mobile applications that follow such a paradigm. EdgeEmu transparently allows Android emulators to participate in large emulated networks while being hosted remotely, in different physical nodes (potentially in a public cloud).

[1] In our paper, the term edge network represents a special case of **ad-hoc networks** that targets mobile devices that are co-located and involved in social interactions.

Fundamental requirements to consider in the design of EdgeEmu include support several Android emulators (only limited by the number of machines) and negligible overhead in terms of latency and bandwidth. To support several emulators, EdgeEmu is designed and developed in a distributed manner. EdgeEmu has two main components. A Client that runs locally on the developer machine that is responsible for creating and modifying the network that is being emulated. A Server that runs on a machine that is used to run emulators that participate in the emulated network. The Server is responsible for managing the Android emulator instances that run on the same machine as the Server. A Client is able to connect to multiple Servers and use the emulators managed by them on the emulated network.

In order to examine the performance of EdgeEmu and show that it fulfills its design requirements, we performed a comprehensive evaluation. Results show that EdgeEmu can support up to 90 Android emulated devices when nine machines with 16 GB RAM are used for running EdgeEmu Servers. The number of supported emulators can vary depending on the number and specifications of the machines running EdgeEmu Servers. Also, results demonstrate that EdgeEmu components induce insignificant overhead in latency and bandwidth tests. In short, EdgeEmu is a new solution for the problem of developing and testing networks of Android emulated devices with negligible overhead in terms of latency and bandwidth while handling a large number of Android devices.

This paper is organized as follows. Section 2 provides a comprehensive study of the current network and cloud computing simulators and emulators, and testing frameworks. Section 3 presents some background information introducing the concept of Android Virtual Device (AVD), and Section 4 presents the architecture of EdgeEmu. A description of the implementation and an experimental evaluation are explained in Section 5 and Section 6, respectively. The paper concludes in Section 7.

2 Related Work

This section covers three areas we consider relevant to this work. First, network simulators/emulators (i.e., systems that help to develop and test network protocols). Then, we compare our work to simulators/emulators aimed at cloud/fog/edge environments; these are centered around simulating/emulating events between multiple devices (and not only network events). Finally, we also analyze other test frameworks, specially mobile application testing frameworks.

2.1 Network Simulators/Emulators

Network simulators are software solutions that can perform tasks in the abstract to demonstrate the behavior of a network and its components, without executing the real/concrete actions of these components or networks.

NS-2 [8] is an open-source, object-oriented TCL (OTcl [9]) script interpreter with a network simulation event scheduler. This network simulator can be used to extensively test new protocol solutions for various network paradigms. NS-2 is feature-rich when considering protocol and network testing, but it supports only a few number of network elements (nodes) that a network can have. The real problem that renders it incapable of supporting EdgeEmu is that NS-2 has no support for mobile application development and testing since it does not support mobile virtual devices.

NS-3 [10] was developed to improve the core architecture, software integration, models, and educational components of NS-2 while maintaining almost all features. The major improvement over NS-2 is using C++ programs or python scripts to define the simulations instead of relying on OTcl as its scripting environment. This made the tool significantly easier to use and build on top of. However, NS-3 still has the same problems as the ones identified on NS-2; it only supports a few nodes and shows no support for mobile application development and testing on top of the simulated network, thus making this tool unsuitable for EdgeEmu.

Other Network Simulation tools, e.g., GloMoSim [11], OMNET++ [12], J-Sim [13], or OPNET [14] provide distinct advantages and disadvantages [15, 16]. Despite any desirable quality all these systems may have, they all fail to provide support for mobile application development and testing on top of the simulated network. This is crucial given the fundamental objective of this work in enabling the development of edge-based applications.

Network emulators are normally available as hardware or software solutions that mimic the behavior of a network to functionally replace it. Network emulators allow 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 [17]. 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. Most network emulation tools do not provide the necessary network characteristics to emulate edge-networks by allowing the network nodes to move within the network. In fact, all the network emulation tools that we analyzed show no support for mobile devices. Nevertheless, some of the tools present interesting solutions to system scalability; below we present the most relevant one.

NetWire [18] is a distributed network emulation system at the physical and MAC layer of the ISO/OSI network model. It follows a client/server architecture 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. Network and node emulation can be spread among multiple workstations to distribute the computational load, by connecting several network servers, drastically improving the system's scalability. However, NetWire has no support for virtual mobile devices or mobile applications being unable to emulate edge-networks since nodes within the network are stationary.

2.2 Fog, Edge and Cloud Simulators

In recent years, a number of simulators have been developed for paradigms such as cloud, edge, and fog computing in the context of, for example, Internet-of-Things (IoT) [19]. EmuEdge [20] is a hybrid emulator and extends Mininet alike systems to emulate edge computing platforms with heterogeneous nodes. CloudSim [21] is a popular simulator for cloud environments, allowing users to simulate cloud-related events such as resource provisioning. EdgeCloudSim [22] improves on CloudSim to target the specific demands of Edge Computing research. In particular, it provides support for the computation and networking abilities inherent to edge computing. CrowdSenSim [23] was developed to simulate Mobile Crowdsensing as it is an appealing paradigm [24].

Compared to current fog, edge, and cloud simulators, EdgeEmu presents a number of advantages when developing Android mobile applications. First, none of the above provide out-of-the-box support for testing Android devices with P2P protocols such as Wi-Fi Direct. Second, these solutions offer support only for simulation and not emulation and, as mentioned for network simulators, simulators provide less support for the design and development of applications. Finally, EdgeEmu provides a GUI that allows users to indicate where a smartphone is located and its path while moving.

2.3 Test Frameworks

There exist several commercial and open-source solutions to provide automated testing and enable the development of mobile applications (for example, MonkeyRunner [25], Appium [26], Expresso [27] and Robotium [28]). All of these tools present distinct advantages and disadvantages between them [29]. However, all these tools show the same problem, i.e., 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 edge-networks.

Termite [30, 31] is an emulation test-bed that provides support for the development and testing of mobile applications. We found Termite to be the only system that has the ability to properly help to develop and test mobile applications running on emulated mobile devices on top of an emulated edge-network, with proper support for node displacement and interactions. Thus, Termite is the system that most closely resembles the one described in this work. However, it presents several downsides when it comes to developing and testing large edge-networks. To overcome the Android SDK limit that emulators can only communicate with up to 15 other emulators spawned in the same physical node, Bruno et al. propose using Android x86 [32]. Using Android x86, emulators can be distributed throughout different physical nodes and communicate with each other. However, using Android x86 images leads to several problems. First, using such images disrupts the normal Android development cycle as applications cannot be easily deployed and debugged with the help of Android Studio. Second, not all emulators (nor supported emulated features) that are available in the Android SDK are available in Android x86. Third, Termite requires a full deployment of a cloud infrastructure platform such as CloudStack [33] or OpenStack [34] to automatically deploy large edge-networks.

3 Android Virtual Devices and Networking

An Android Virtual Device (AVD) [35] defines the characteristics of an Android phone, tablet, Wear OS, Android TV, or Automotive OS device that is virtualized through an Android Emulator. In this particular work, an AVD configuration represents any Android mobile device. AVDs contain the hardware profile, system image, storage area, skin, and other properties that represent the Android device that developers may want to emulate. With this information, developers are able to create multiple unique Android emulators with the features described in their configuration.

When using the Android SDK, Android emulators run inside an isolated network inside the developer's machine. Each emulator instance runs behind a virtual router/firewall service that isolates it from the development machine, network interfaces, and from the Internet. According to the official Android Emulator networking documentation [36], there is a virtual router for each AVD instance that manages the `10.0.2.0/24` network address space; therefore, all addresses managed by the router are in the form of `10.0.2.N`, where `N` is a number between 0 and 255. Addresses within such address space are pre-allocated by the emulator/router [36]. In this paper, Android emulators are also referred to as emulators hereafter.

So, consider a scenario in which we have an application running inside an AVD emulator instance (e.g., a smartphone app) communicating with some local service, listening on port 90 on IP `127.0.0.1` (in the developer's local machine). Note that both the developer machine's 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' 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 AVD emulator instance as a process that can be accessed via a pair of control ports. This pair of control ports (on the developer machine) correspond to: i) a default control port, and ii) an `Android Debug Bridge (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 AVD emulator instance has a control port and an `adb` client port pair. These ports are sequentially attributed to each AVD emulator instance until the maximum port number is 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 AVD emulator instance and not to the mobile application that might be running inside; such ports are meant only to detect and configure the AVD emulator instance. The communication between any process on a developer machine network environment and an application running inside an AVD 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 process, running on the developer machine, to an application, acting as a server, inside emulator A that is listening on port 9001. 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 this end, we first need to set a

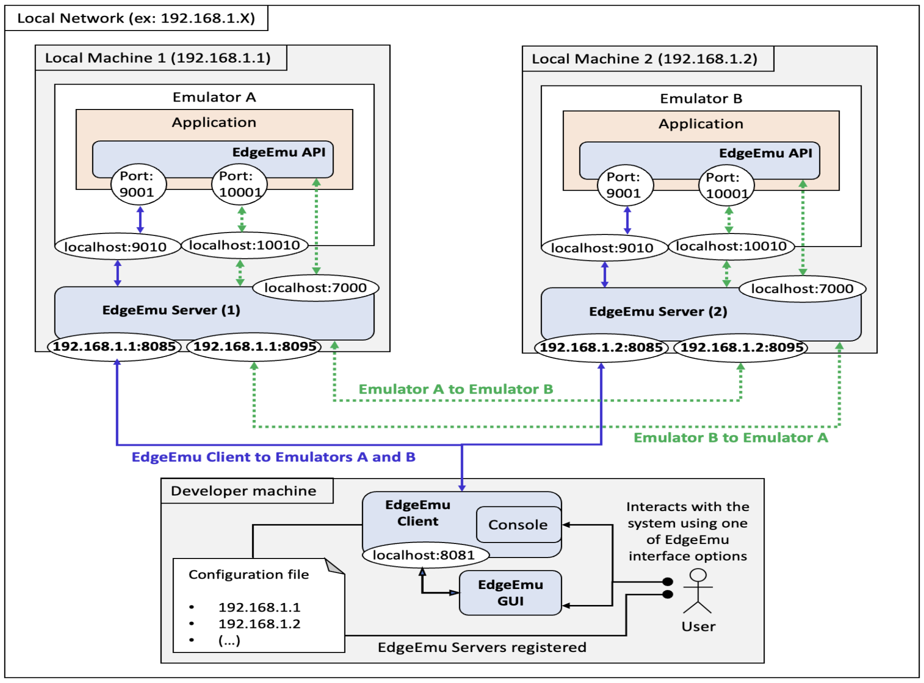

Fig. 1. EdgeEmu components and their interactions.

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`. The port redirection command instructs 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 emulator's network environment). Assuming X is 9010 and Y is 9001, the client application process 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 emulator's localhost network.

4 EdgeEmu Emulation

We designed EdgeEmu to allow developers to develop and test applications that require large edge-networks. To that end, EdgeEmu is able to use emulators that are running across multiple machines (see Figure 1).

As mentioned before, EdgeEmu consists of two main components: a Client and a Server. The **EdgeEmu Client** runs from a console window on the developer machine where the emulated network is created and modelled. User interactions with EdgeEmu can also be performed through this Client. The user is able to choose between two interface options: the text-based console where system interactions are done using written commands, or a GUI where system interactions are performed via interactive menus and options (see Section 5 for details).

An **EdgeEmu Server** needs to be running on each machine where we want to run emulators. In order for an EdgeEmu Client to access an EdgeEmu Server (and the emulators managed by it), the user must register them. This consists on writing the local IP addresses of the machines where EdgeEmu Servers are running on a configuration file provided to the EdgeEmu Client (see Figure 1). This configuration file is a simple plain text file. Each local IP address of the EdgeEmu Servers' machines should be written in a separate line of the file. For instance, in Figure 1, the local IP address of machine 1 is `192.168.1.1`. This IP address is automatically discovered by the EdgeEmu Server and can be used for accessing to EdgeEmu Server. Clearly, the user can assign arbitrary IP addresses to the EdgeEmu Servers via EdgeEmu Server-side plain text configuration file.

Thanks to the EdgeEmu Server, EdgeEmu is able to support a large number of emulators distributed across multiple machines. In EdgeEmu, emulators are identified using the machine's local IP address where they are running and two pairs of network addresses and port numbers. For example, in Figure 1, 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`). These addresses are set by the EdgeEmu Servers. These addresses enable two types of interactions between EdgeEmu components: *Control Messages* and *Data Messages*.

Messages are sent from the EdgeEmu Client to the emulators following instructions given by the developer. These messages are called *Control Messages* (blue/solid arrows between the Client and the EdgeEmu servers 1 and 2) and are sent in two cases: i) when nodes (within the emulated network) move close to others, or ii) when P2P groups are formed between nodes (also within the emulated network). *Control Messages* contain information that allows the triggering of P2P events inside an application (e.g., using the Wi-Fi Direct API provided by Android). One such event is the creation of a P2P group with other target emulators. When such an event occurs, an application creates a socket connection with other group members, allowing them to communicate through *Data Messages* (green/dotted arrows). For example, taking into account Figure 1, emulator A connects with the target emulator B. This connection is then redirected to the application (running in emulator B) and received by a socket also opened by the EdgeEmu API on port 10001.

As illustrated in Figure 1, the EdgeEmu Client connects with each registered EdgeEmu 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 EdgeEmu Server but can be changed to any other port value chosen by the user through the EdgeEmu Server-side plain text configuration file. With these connections, the EdgeEmu Client is then able to discover and use the emulators that are running on each EdgeEmu Serv-

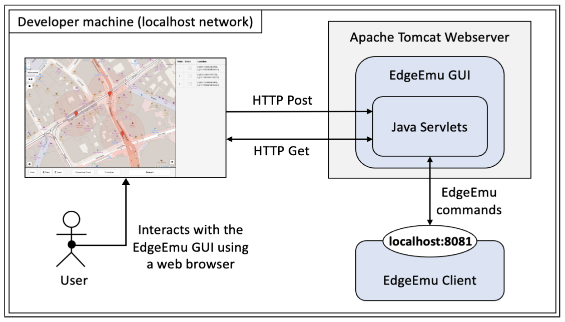

Fig. 2. EdgeEmu GUI implementation.

er machine. These connections are used by the EdgeEmu Client to send the *Control Messages* to emulators using the EdgeEmu Servers as intermediaries.

As already mentioned, emulators can communicate with each other by using EdgeEmu to create a socket connection with the group member (the socket connection is created using the addresses of each emulator, provided within the control message information). This connection is performed with the EdgeEmu Server(s) of each emulator as intermediaries. In Figure 1 we can see this and the addresses used by looking at the green/dotted arrows.

Lastly, it is worth mentioning that EdgeEmu easily allows the user to manage the emulators' life cycle (create, destroy, start, stop and install, and start applications) from the EdgeEmu Client. This is crucial for EdgeEmu to support several Android emulators as it allows a user to create and manage a large number of them (and the applications running inside) distributed across multiple machines from a single control point.

5 Implementation

Most components in EdgeEmu were implemented using Java version 17. We use this language to guarantee that it can easily run on any platform and that it is easily extendable. In addition, as Android libraries and applications are largely written in Java [37], it was a natural choice.

The EdgeEmu GUI runs on top of an Apache Tomcat Server version 9 and the interface logic is implemented using JavaScript ES6. We opted for JavaScript as it allows the interface logic to run across any modern web browser (Google Chrome, Firefox, Safari, etc.). EdgeEmu supports both Google Maps API [38] and OpenStreetMap API [39] and gives users an option to select their preference. We chose Google Maps due to its extensive documentation and features and OpenStreetMap as it is free.

The GUI performs two types of communications with the EdgeEmu Client: i) requesting data (e.g., requesting emulators to be used on the emulated network), and ii) sending data (e.g., sending Control Messages to the target emulators or commands to start new emulators). These communications (illustrated in Figure 2) 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 EdgeEmu commands (similar to those when a user uses the console interface). The commands are then sent from the Servlets to the EdgeEmu Client (through a socket connection on `localhost:8081`) to be processed.

6 Evaluation

In this section, we compare EdgeEmu with Termite, a previous existing open-source system (described in Section 2). Looking both at Termite and EdgeEmu's architectures, we can identify that the main difference is the EdgeEmu Server and EdgeEmu Client components (both used to relay messages). These components, both in EdgeEmu, completely change how messages are exchanged within the system: both when we consider the communication protocols between the EdgeEmu Client and the emulators, and also between emulators. In fact, in Termite, messages are exchanged directly from the sender emulator to the receiver. In EdgeEmu, such communication is not direct because the EdgeEmu Server acts as a middle point responsible for redirecting messages sent from the EdgeEmu Client to the target emulators and messages exchanged between emulators.

To compare both systems and focus on the most differentiating aspects, we developed two tests: i) the Ping Test, a latency test (see Section 6.2) that measures the time it takes to send a ping message from the EdgeEmu Client to the emulators and receive a response; ii) the File Sharing Test (see Section 6.3), a bandwidth test that measures the time it takes to send a large message from one emulator to another and receive a response.

We also conduct additional experiments using latency and bandwidth tests where an increasing number of emulators is either deployed locally or remotely (see Section 6.4). First, we run both tests on Termite and EdgeEmu while running locally (i.e., all system components and emulators run on the same developer machine). Then, we execute the same tests on EdgeEmu, this time running all the necessary AVD emulator instances remotely. This means that on one machine we run the EdgeEmu Client while the necessary emulators run on multiple remote machines. On each remote machine, there is an instance of the EdgeEmu Server. The number of emulators running on each machine varies according to the test being performed.

6.1 Testing environment

When performing the tests locally, all elements of both Termite and EdgeEmu execute on a single cluster node. All cluster nodes run Ubuntu 18.04, and are equipped with an Intel Core i5-4460 3.20GHz (Quad-core) CPU, with 16 GB RAM. All the used emulators correspond to a Pixel2 Android phone running Android 5.1 with API 21. This Android version and API allow us to cover more than 99% of the existing smartphones [40]. When performing tests where the AVD emulators' instances are distributed throughout different cluster nodes, the EdgeEmu Client executed in a local MacBook Pro with macOS Catalina 10.15.7, featuring a (Dual-Core) Intel Core i5 2,7 GHz CPU (4 threads), with 8GB of RAM.

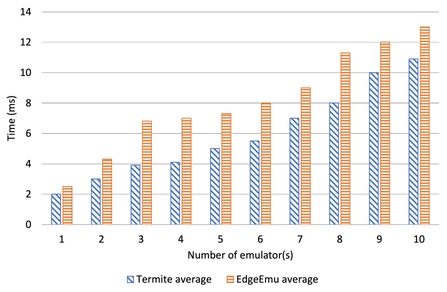

Fig. 3. Ping test local results.

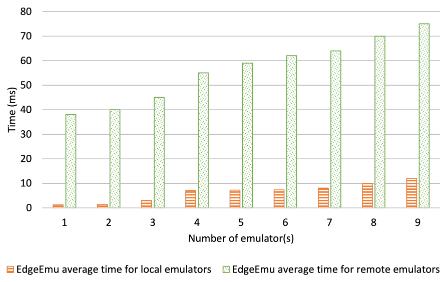

Fig. 4. Ping test remote results.

We never run more than ten emulator instances simultaneously on a single cluster node. This was done due to memory constraints as Android emulators have high memory requirements, and over committing memory rapidly slows down emulators' response times. In fact, running 10 emulators consumes around 13GB of RAM with a 40% of CPU usage. Thus, with a cluster node, having 16GB of RAM (14GB usable), running more than ten emulators starts to severely impact system speed and responsiveness.

6.2 Ping Test

The Ping test was developed to evaluate EdgeEmu Server's latency impact when messages are exchanged between an EdgeEmu Client and the emulators. The test consists of measuring the time it takes to send a single message from the EdgeEmu Client component to an increasing number of AVD emulator instances. Thus, we created a test script that would send a message to each emulator and measured the time that each message took to be received and a response being sent. Starting with one emulator instance, we executed the test script until the variance of time values obtained becomes less than 10%. In order to ensure the validity of results, we performed the test script at least ten times. 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.

As explained before, the test was performed in two different scenarios. First, we executed the tests running all system components and the emulators locally (i.e., in a single machine). Then, we performed the same test but instead of increasing the number of emulators that run locally, we increased the number of remote machines being used with each machine running a single emulator.

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

For the tests conducted with each emulator running in different remote cluster nodes, the results show (see Figure 4) there is a performance impact when

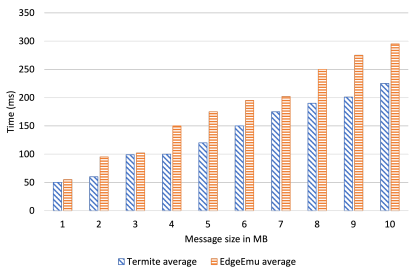

Fig. 5. File Sharing test local results.

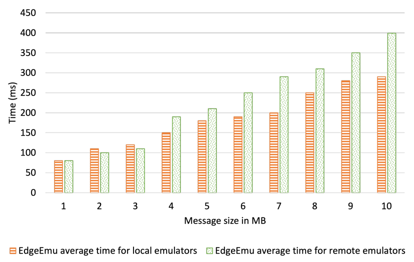

Fig. 6. File Sharing test remote results.

compared with the local EdgeEmu 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 EdgeEmu Client and the cluster nodes. This is expected and we believe that 40-70 ms of latency does not compromise the usability of the system as many real systems that operate over the Internet also include similar latencies.

6.3 File Sharing Test

We now evaluate EdgeEmu bandwidth impact when 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 MBs) from one emulator to another (when they are part of a P2P group). To that end, we create an Android mobile application (running in emulators) that detects the creation of the P2P group and automatically sends the file to the other group member (i.e., the other emulator). The application then measures the time between sending the file and receiving a response from the other group member. In order to trigger this P2P event between the two emulators, we run a script using EdgeEmu commands that emulate the creation of the P2P group between two emulators running the app previously mentioned. We ran this test script until the variance of time values obtained was less than 10%. To respect the reliability of the results, the test script was executed at least 10 times. Similar to the Ping Test, this test was performed in two different scenarios. After all tests are finished, we calculate the average time value for each test.

Results obtained for the File Sharing test on Termite and EdgeEmu with both emulators running on the same machine can be seen in Figure 5. The results show that EdgeEmu 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 MB). We conclude that EdgeEmu presents a similar performance as Termite on this type of communication when emulators are running on the same machine.

For the tests conducted on EdgeEmu with each AVD emulator instance running on different cluster machines, the results show (see Figure 6) that there is a performance impact when compared with the local EdgeEmu test results: an increase of 20-109 milliseconds per MB when compared with the results obtained on EdgeEmu 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 cluster machines over the network. Nevertheless, we consider this overhead to be completely normal as it is mostly due to network communication latency and not due to EdgeEmu. Furthermore, we expect real applications to send small to medium size files as sending large files through a P2P connection (Wi-Fi Direct) is not recommended due to the unstable nature of this type of communication (devices come in and out of proximity of each other depending on their speed).

In conclusion, EdgeEmu Server has a negligible impact on the time that an emulator takes to send and receive messages from and to another emulator. 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 EdgeEmu Server and redirected to the destination (to the Server that is managing the target emulator). When both emulators are running on the same machine the performance impact is low; the impact is higher when the emulators are running on different machines. However, we believe that in both cases the performance impact is acceptable for the communication paradigm we are using and we believe that EdgeEmu presents an acceptable performance difference when compared to Termite.

6.4 Number of Emulators

The number of AVD emulator instances we can use within the emulated network running the application we wish to develop and test has an obvious impact on EdgeEmu performance. 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 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 developer laptops and desktops.

On the contrary, with EdgeEmu, one can now create and manage all AVD emulator instances directly from an EdgeEmu Client. This removes the need to use Android AVD Manager or user-made scripts to manage the emulators. Using EdgeEmu also provides 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 create the emulated network on one machine and run the AVD emulator instances distributed across other machines. As a result, users can create much larger emulated networks.

Local Deployment: This test evaluates the time that it takes to launch an increasing number of AVD emulator instances running a test application using both Termite and EdgeEmu. The test consists of measuring the time it takes to launch a selected number of emulator instances, installing an application on each emulator and running it.

To perform this test on Termite we created a simple script file using Termite commands to perform the same actions of launching the emulators, installing the applications, and running them. Starting with one emulator instance to a

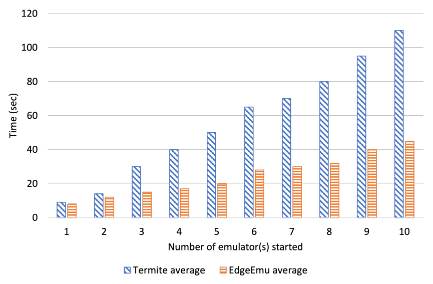

Fig. 7. Local Deployment results for Termite and EdgeEmu.

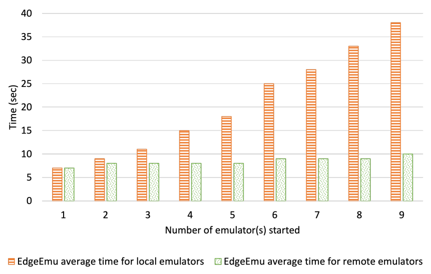

Fig. 8. Startup time for local and remote emulators using EdgeEmu.

maximum of 10, the test runs until the variance between time values obtained was less than 10%. We ran the test script at least 10 times to ensure the validity of the results. This test was performed on a single cluster machine. Results for this test are shown in Figure 7. By looking at the plot it is possible to see that EdgeEmu presents a significant improvement in 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 Termite starts the emulator instances sequentially while EdgeEmu starts all emulator instances in parallel. We can see this by considering that the time improvement when we use EdgeEmu is greater when the number of emulators used increases.

Distributed Deployment: From the previous test results we were able to show how EdgeEmu speeds up the deployment of the 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 on the same machine.

To truly show how EdgeEmu behaves with an increasing number of emulators, we developed a similar application to the previous one but now each emulator instance runs on a different cluster machine. To perform the test, we used a similar script file as the one created for the Local Deployment test, but now we distribute the emulators across multiple cluster machines. This script file was loaded on EdgeEmu Client and the emulators started on remote machines.

First, we started by measuring the time it takes to deploy a single emulator per machine, installing the template application and starting it. Then, we perform the same test but this time we deployed 10 emulators instances per remote machine. With 9 cluster machines, we were able to launch a total of 90 emulators, a drastic improvement over the maximum of 16 emulators that one can run when using Termite.

In Figure 8 we show the values obtained when starting one emulator across the 9 cluster machines, and compare these values to those obtained on the Local Deployment test, where we started the same amount of emulators (using EdgeEmu) on a single cluster machine. We can see that with the distributed approach, the time it takes to start one emulator on a single machine and start

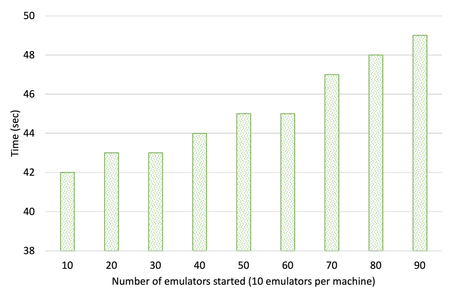

Fig. 9. The time that it takes to start 10 emulators across the 9 cluster machines.

9 emulators across 9 different machines is approximately the same (the slight difference is because of the network delay between EdgeEmuClient machine and cluster machines). This happens due to the fact that the EdgeEmu Client processes/sends the start commands to the EdgeEmu Servers on multiple machines at the same time. Thus, with EdgeEmu it is possible to explore the inherent parallelism of a distributed system.

This parallel processing is again shown on the results obtained when starting 10 emulators across each one of the 9 cluster machines (see Figure 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).

Finally, by looking at Figure 9 and comparing the results obtained to those that we presented on the Local Deployment test, we can see that starting 90 emulators takes less time than to start 10 emulators on Termite (using Android Studio which starts the emulators sequentially). With these results, we can easily conclude that EdgeEmu offers better performance when compared to Termite.

7 Conclusion

On the one hand, network simulation and emulation tools available today do not provide sufficient support to develop and test edge-based applications. On the other hand, existing tools that emulate edge-based networks lack the necessary network layer where tests must be performed. Termite, the closest system to EdgeEmu, does not perform well with an increasing number of Android emulators.

This work presents EdgeEmu. It proposes a 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.

References

1. W3Techs, “Gsma intelligence,” last accessed February 2023. [Online]. Available: https://www.gsmaintelligence.com

2. STATISTICA, “Mobile internet usage worldwide - statistics and facts,” last accessed February 2023. [Online]. Available: https://www.statista.com/topics/779/mobile-internet
3. S. Baek, J. Ahn, and D. Kim, “Future business model for mobile cloud gaming: the case of south korea and implications,” *IEEE Communications Magazine*, pp. 1–7, 2023.
4. S. Kaisar, J. Kamruzzaman, G. Karmakar, and M. M. Rashid, “Decentralized content sharing in mobile ad-hoc networks: A survey,” *Digital Communications and Networks*, 2022.
5. Wi-Fi Alliance, “Portable wi-fi that goes with you anywhere,” last accessed February 2023. [Online]. Available: https://www.wi-fi.org/discover-wi-fi/wi-fi-direct
6. Google, “Android studio the official integrated development environment (ide) for android app development,” last accessed February 2023. [Online]. Available: https://developer.android.com/studio
7. Google, “Android standard development kit,” last accessed February 2023. [Online]. Available: https://developer.android.com/studio
8. NS-2, “The network simulator - ns-2,” last accessed February 2023. [Online]. Available: http://nsnam.sourceforge.net/wiki/index.php/Main_Page
9. D. Wetherall, “Otcl - mit object tcl,” last accessed February 2023. [Online]. Available: http://otcl-tclcl.sourceforge.net/otcl/
10. NS-3, “Ns-3 network simulator,” last accessed February 2023. [Online]. Available: https://www.nsnam.org
11. L. Bajaj, M. Takai, R. Ahuja, K. Tang, R. Bagrodia, and M. Gerla, “Glomosim: A scalable network simulation environment.”
12. OMNeT++, “Omnet++ discrete event simulator,” last accessed February 2023. [Online]. Available: https://omnetpp.org
13. J-Sim, “J-sim network simulator,” last accessed February 2023. [Online]. Available: http://www.kiv.zcu.cz/j-sim/
14. OPNET Optimum Network Performance, “Opnet network simulator,” last accessed February 2023. [Online]. Available: http://opnetprojects.com/opnet-network-simulator/
15. 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.
16. 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.
17. 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.
18. 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.
19. A. Markus and A. Kertesz, “A survey and taxonomy of simulation environments modelling fog computing,” *Simulation Modelling Practice and Theory*, vol. 101, p. 102042, 2020, modeling and Simulation of Fog Computing. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1569190X1930173X
20. Y. Zeng, M. Chao, and R. Stoleru, “EmuEdge: A Hybrid Emulator for Reproducible and Realistic Edge Computing Experiments,” in *2019 IEEE International Conference on Fog Computing (ICFC)*, Jun. 2019, pp. 153–164.

21. R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw. Pract. Exper.*, vol. 41, no. 1, p. 23–50, Jan. 2011. [Online]. Available: https://doi.org/10.1002/spe.995
22. C. Sonmez, A. Ozgovde, and C. Ersoy, "Edgecloudsim: An environment for performance evaluation of edge computing systems," in *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*, 2017, pp. 39–44.
23. C. Fiandrino, A. Capponi, G. Cacciatore, D. Kliazovich, U. Sorger, P. Bouvry, B. Kantarci, F. Granelli, and S. Giordano, "Crowdsensim: a simulation platform for mobile crowdsensing in realistic urban environments," *IEEE Access*, vol. 5, pp. 3490–3503, 2017.
24. R. K. Ganti, F. Ye, and H. Lei, "Mobile crowdsensing: current state and future challenges," *IEEE Communications Magazine*, vol. 49, no. 11, pp. 32–39, 2011.
25. Google, "Monkeyrunner user guide," last accessed February 2023. [Online]. Available: https://developer.android.com/studio/test/monkeyrunner
26. JS Foundation, "Appium automation for apps," last accessed February 2023. [Online]. Available: http://appium.io
27. Open Source, "Expresso framwork," last accessed February 2023. [Online]. Available: https://developer.android.com/training/testing/espresso
28. RobotiumTech, "Robotium user scenario testing for android," last accessed February 2023. [Online]. Available: https://github.com/RobotiumTech/robotium
29. S. Gunasekaran and V. Bargavi, "Survey on automation testing tools for mobile applications," *International Journal of Advanced Engineering Research and Science*, vol. 2, no. 11, pp. 2349–6495, 2015.
30. 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.
31. N. Santos, P. Ferreira, and R. Bruno, "Termite: Emulation testbed for encounter networks," last accessed February 2023. [Online]. Available: https://nuno-santos.github.io/termite/index.html
32. Android-x86, "Android-x86 - run android on your pc," last accessed February 2023. [Online]. Available: https://www.android-x86.org/
33. Apache CloudStack, "Apache cloudstack - open source cloud computing," last accessed February 2023. [Online]. Available: https://cloudstack.apache.org/
34. "Open Source Cloud Computing Infrastructure," last accessed February 2023. [Online]. Available: https://www.openstack.org/
35. Google, "Create and manage virtual devices," last accessed February 2023. [Online]. Available: https://developer.android.com/studio/run/managing-avds
36. Google, "Set up android emulator networking," last accessed February 2023. [Online]. Available: https://developer.android.com/studio/run/emulator-networking
37. Pam, "Java Mobile Applications Development," Nov. 2021. [Online]. Available: https://www.webiotic.com/java-mobile-applications-development-what-you-need-to-know/
38. Google, "Google maps platform documentation," last accessed February 2023. [Online]. Available: https://developers.google.com/maps/documentation
39. S. Coast, "OpenStreetMap wiki," last accessed February 2023. [Online]. Available: https://wiki.openstreetmap.org/wiki/Main_Page
40. Google, "Android api levels," last accessed February 2023. [Online]. Available: https://apilevels.com/