UNIVERSITY OF TARTU
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
Institute of Computer Science

MIHKEL VISNAPUU

Device-to-Device Mobile Gaming

Bachelor Thesis (6 EAP)

Supervisor: Huber Flores, MSc
Co-supervisor: Satish Narayana Srirama, PhD

TARTU, 2015

Abstract

Device-to-Device(D2D) mobile gaming is a new trend which is emerging as a result of the increasing advances in mobile devices and social network interaction with mobile peers. As these games are played between players in proximity, it is possible to take advantage of computational offloading to balance the load of these applications.

Smartphone games can be instrumentalized with computational offloading mechanisms in order to save energy and increase response time of the applications. In this context, remote cloud and D2D offloading has been proposed.

It is well known that low latency is preferable to high latency in the communication when offloading, and as a result, D2D offloading is more suitable than remote cloud.

However, the idea of offloading to a nearby device is not feasible in practice, because a user may not be willing to process the task from another device. This can be clearly seen as processing a task from another device does not represent a gain but rather a loss in resources for the device that executes the task.

In this thesis, we investigate a new perspective, in which a device is not requested to process a task, but it is alleviated from processing one task that another device has already processed.

To achieve this purpose, we develop a framework and a case study. Based on the result of the validation, we found out that it is possible to balance the execution load of an application between nearby interconnected devices.

Keywords:

Android, Code offloading, Bluetooth

Contents

List of Figures

LIST OF FIGURES

1

Introduction

1.1 Introduction

1.1.1 Motivation

Global smartphone usage has drastically increased in recent years and it is estimated that by 2018 one third of consumers worldwide will be using them [1]. This is due to the increase in inexpensive smartphones coming to the market, which increases the need for applications to use less resources to accommodate low-end devices. Also because smartphones have batteries that are limited by size and thus capacity, it is extremely important to handle energy consumption optimally. It is common to charge the battery daily. Code offloading is an approach that could foster better energy saving for the smartphones resources (1).

The proliferation of smartphone applications is on the rise, in particular mobile games, which already have PC-like features. D2D mobile games is a trend that is emerging as a result of this sophistication. While code offloading can be utilized to delegate resource intensive tasks, it can also be utilized to balance the execution load of using mobile applications when they are connected in proximity.

Our hypothesis is that intermediate results can be shared between devices. For instance, in the case of 3D mobile games, the 3D models(.obj, .x3d, .3ds, etc.) are large and require heavy computational processing to build. Lets imagine a multiplayer mobile game that allows the user to take a video of a room, process it and create a 3D model that will be used to create a new level. When this game is played with other nearby devices, they too would need to get the model in order to visualize it in-game. Instead of going through the process of making the

[1] http://www.emarketer.com/Article/2-Billion-Consumers-Worldwide-Smartphones-by-2016/1011694

video and creating the 3D model, it is possible to share the already processed 3D model and so decreasing the processing load for the device.

1.1.2 Contributions

A framework to support D2D offloading was developed. The framework follows a master/slave model, in which the master device gives the slaves offloading tasks. Java reflection is used to offload to other devices. Also a simple 2D battle game was developed to validate the framework.

1.1.3 Outline

Chapter 2: discusses the state of the art for code offloading.
Chapter 3: provides the problem statement for the thesis. In particular, we look at the possibility of code offloading for D2D mobile games.
Chapter 4: describes the contribution of the thesis. This section includes the discussion of the developed framework as well as the game that serves as the use case.
Chapter 5: provides the conclusions for the thesis.
Chapter 6: describes the future research directions
Chapter 7: is the abstract in Estonian.

2

State of the Art

2.1 Mobile Cloud Computing

Mobile computing is a technology that allows the device to transmit data without having to be connected to a fixed physical link [2]. For example being able to read the news and stay connected with friends and family while on the move is possible thanks to mobile computing.

Cloud computing is a technology that allows the user to consume services, which follows an utility model (2, 3, 4, 5). It provides virtually infinite processing capabilities as servers to the end users (2). These servers are accessed by the users using thin clients (6).

Because mobile devices are constrained by limited storage, processing capabilities, memory, battery etc., connecting them to the cloud enables augmenting these constraints (7, 8, 9, 10). The most prominent technique to empower the mobile devices with cloud power is code offloading and we will discuss this in more detail in the upcoming sections.

2.2 Code offloading

Code offloading refers to a technique in which a computational task is transferred from one place to another and then processed there (3, 7, 8, 11). As long as both execution environments are the same, a computational task can be transferred between them. A general schema for code offloading is shown in Figure 2.1.

The primary purpose of code offloading is to decrease the energy usage in the device as this is one of the biggest constraints of mobile devices today. By diverting energy consuming

[2] http://www.doc.ic.ac.uk/ nd/surprise_96/journal/vol4/vk5/report.html

processes from client to server, this technique allows balancing and maintaining energy usage in the client. It is a necessity for code offloading to take less or equal amount of time to execute, otherwise it would make the application unresponsive and would drive the user away. Offloading is beneficial when large amounts of computation is needed with relatively small amount of data used for connection (4).

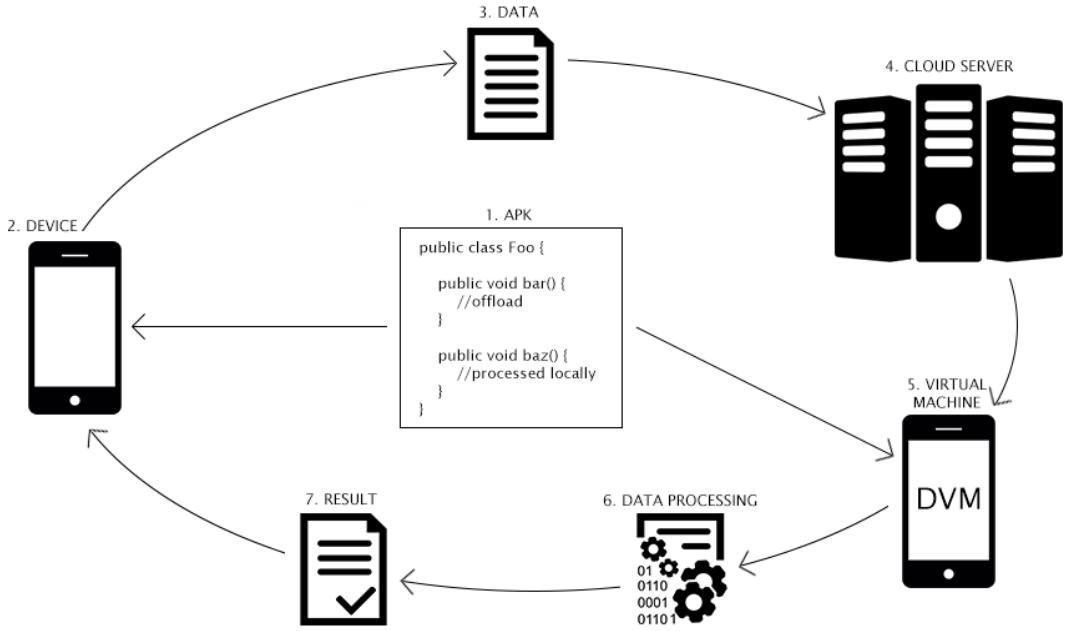

Figure 2.1: General code offloading schema

Figure 2.1 shows the traditional model for code offloading. The application(1), is installed in both the device(2) and Dalvik *V*irtual *M*achine (VM)(5) located in the cloud server(4). When the bar method is called, the offloading framework in the device sends the necessary data(3) to the Dalvik VM, that then executes the method(6) and sends the result(7) back to the application in the device. This means that the device got the result for the method, without actually executing it itself.

2.3 Technologies and Implementations

There are different technologies and implementations that have been made for offloading. In this section we will be looking at the most prominent of these.

2.3.1 Java Reflection

The Reflection API has been included in Java since version 1.1 [3]. It allows to examine or modify the runtime behaviour of applications [4].

The Snippet 1 shows a simple example of Java reflection, where the method localFoo is executed from the class Test by calling the method foo. In order to execute localFoo, the method is first captured using getMethod function, which requires the method name and parameter types as parameters. As localFoo does not require any parameters, null values are given to getMethod and invoke calls. The invoke function then executes the method localFoo.

Snippet 1 Example of Java reflection implementation

```
public class Test {
    public void localFoo() {
        //do something
    }

    public void foo() {
        Class<?> paramTypes = null;
        Object[] paramValues = null;
        Method method = Test.getClass().getMethod(
            "localFoo",
            paramTypes
        );
        method.invoke(Test, paramValues);
    }
}
```

Java Remote Method Invocation (RMI) —RMI uses object serialization to assemble and disassemble parameters and does not truncate types, supporting true object-oriented polymorphism [5]. Java RMI system allows an object running in one Java VM to invoke methods on an object running in another Java VM [6].

Java Remote Procedure Call (RPC) —RPC follows a client-server model [7] where the client can call for the execution of the method in the server. Unlike RMI, the client does not have the

[3] http://docstore.mik.ua/orelly/java-ent/jnut/ch14_01.htm
[4] http://docs.oracle.com/javase/tutorial/reflect/index.html
[5] http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html
[6] https://docs.oracle.com/javase/tutorial/rmi/
[7] http://www.cs.cf.ac.uk/Dave/C/node33.html

code to execute, but the reference to the code which resides in the server.

2.3.2 .NET framework

.NET Framework is a software framework developed by Microsoft [8]. Among other objectives it is designed to remotely execute code [9]. It consists of the Common Language Runtime (CLR) and the .NET Framework class library [10]. The CLR is the execution engine that handles running applications by providing memory management and other system services [11]. CLR is used by implementations such as MAUI (7).

2.4 Computational Offloading Frameworks

The idea behind computational offloading, which is also known as cyber foraging (12), is to dynamically augment the computational and storage capabilities of mobile devices by taking advantage of opportunistically discovered servers in the environment (13).

There have been many breakthroughs in code offloading over the years as it is a subject that has been researched for over a decade. In this section we will be looking at the most prominent of these solutions.

2.4.1 Cloudlets

Cloudlets are decentralized and widely-dispersed Internet infrastructures whose compute cycles and storage resources can be leveraged by nearby mobile computers (6). The purpose of them is to bring cloud closer to mobile devices, meaning that the connection could be established by Wireless LAN instead of WAN. By doing this, the delays in connection can be brought down.

The connection between mobile devices and cloudlets can be viewed as a client server relationship. In this sense the mobile devices are thin-clients as the bulk of data processing occurs on the server (cloudlet) [12].

Using cloudlets takes the burden away from the programmer to modify the application for offloading purposes.

[8] https://msdn.microsoft.com/en-us/library/ff361664(v=vs.110).aspx
[9] https://msdn.microsoft.com/en-us/library/vstudio/zw4w595w(v=vs.100).aspx
[10] https://msdn.microsoft.com/en-us/library/w0x726c2(v=vs.110).aspx
[11] https://msdn.microsoft.com/en-us/library/hh425099(v=vs.110).aspx
[12] http://www.webopedia.com/term/t/thin_client.html

2.4.2 Mobile Assistance Using Infrastructure

MAUI is a system based on .NET framework, that at runtime uses an analyser to decide what code should be executed remotely, determined from performance and energy standpoints.

Annotations are used to let the application developer and analyser determine which methods and/or classes can be offloaded (7). The developer marks the ones that can be offloaded and the framework determines whether it should be offloaded. Local execution is used as a fallback in case the remote execution is not possible for various reasons.

The tests that were run indicate that MAUI has the capabilities to reduce processing and energy usage in mobile devices (7).

2.4.3 ThinkAir

Similar to MAUI, ThinkAir (8) provides method-level computation offloading using annotations. However it addresses MAUIs lack of scalability by creating VMs of a complete smartphone system on the cloud (8).

On first encounter, the analyser that is used to determine whether to offload or not, takes into account current environmental parameters and starts collecting data for future usage. In later stages, the collected data is used to determine where to execute the method. Java reflection is used for offloading (8).

ThinkAir provides an efficient way to perform on-demand resource allocation and exploits parallelism by dynamically creating, resuming, and destroying VMs in the cloud when needed. Parallel execution is exploited by either using multiprocessor support or splitting the work among multiple VMs. By doing this it was possible to reduce the execution times and energy consumption of applications, compared to non-parallel executions (8).

2.4.4 COMET

Code Offloaded by Migrating Execution Transparently (COMET) (9) is a system that focuses on improving the speed of computation. In order to achieve this, they introduced Distributed Shared Memory (DSM) to offloading. By doing this they have succeeded in developing an offloading engine that fully supports multi-threaded computations. As such COMET is more focused on not what to offload but how to offload (9).

By offloading computationally heavy tasks over WiFi, the system has on average managed to decrease battery consumption of the applications. It is also noted that due to 3G characteristics, offloading via 3G usually ends up consuming more energy as opposed to executing locally. This is however taken care of by the decision engine, when latency is high and bandwidth is limited, the tasks are run locally.

2.4.5 Evidence-aware Mobile Code Offloading

EMCO (14) follows the traditional offloading model. However it also adds data analysing and a cache. By gathering data from users and analysing it with Evidence analyser, it is possible to determine what, when and where to offload more optimally (5, 14).

Cache allows reusing the result for a given code that is often called, thus lowering the execution time. It can also store results in client side, if it is determined to be reused again or later in the applications execution. Initially the advantages of this approach would be comparable to other proposals, however as it takes advantage of crowdsourcing, it should show it's true potential over time (14).

2.5 Device-to-Device (D2D) Communication

As demonstrated by previous works offloading to remote cloud is feasible. However the latency issues in communication is still a major drawback. Another proposed solution is to offload to nearby devices, which are in a low-latency networks. Latency is the time taken to send data between two points in a network, a low-latency network is where this time taken is minimized [13].

Mainly two connection protocols are used in D2D mobile clouds - WiFi and Bluetooth. The main downside of Bluetooth is its limited range (~10 m), compared to WiFis range of around 100 meters (15). However the upside of Bluetooth is that the power consumption is low (16).

It is also important for the participating devices to have incentive to share their resources with other devices and there needs to be a mechanism to prevent 'free riding' (15).

2.5.1 Context-Aware Hybrid Computational Offloading

The main idea behind dynamic D2D infrastructure is to create a dynamic infrastructure of multiple mobile devices in proximity to share the load of processing heavy computational tasks (1).

[13] http://www.cl.cam.ac.uk/teaching/0708/AddTopics/Low-Latency-Networking.ppt

The D2D infrastructure is created by transforming nearby devices into servers, which can process offloading tasks from other devices.

The system combines features from cloudlets and code offloading models, by offloading to cloud and relying on D2D communication to foster computational offloading in proximity. It is adaptive to the applications context, as the system can decide whether to offload to cloud or to a D2D infrastructure nearby. (1)

2.5.2 Serendipity

Serendipity (11) is a system, that enables mobile devices to remotely access computational resources of other mobile devices (11). Benchmarks are ran by profilers to gather data about the devices capabilities. When two devices encounter, they first exchange metadata, which also includes the data from profilers (11). This data is used to determine if it is feasible to offload from execution time and energy consumption standpoints.

However as mobile devices have limited energy, the user might not want to share the energy they have. Serendipity proposes that the reasonable solution would be for each device to last as long as possible while still timely finishing their tasks (11). To battle this they use an algorithm proposed in (17).

2.5.3 Hyrax

Hyrax uses a cluster of mobile devices as resource providers and have succeeded in showing the feasibility in such a mobile cloud (15). A modified Hadoop [14] is transplanted into Android so that these devices can act as PCs to deploy a real cloud computing system (18). WiFi is used to establish connection with nearby devices (19).

2.6 Summary

In this section we explained what is computational offloading, how it can be implemented and briefly looked at the current solutions provided. From the works done, it can be seen that offloading succeeds in being able to improve performance and decrease power consumption. However it is also stated that the tests are mainly done in controlled environments and because of this in most cases computational offloading is actually counterproductive in real-world scenarios (1).

[14] https://hadoop.apache.org/

3

Problem Statement

In the previous section, we explored current solutions for computational offloading. Most of the frameworks discussed take advantage of remote cloud, which has its advantages and disadvantages. One of the biggest disadvantage is that the connection established with cloud servers can suffer from high-latency. In this section we raise the question, how is it possible to acquire computational resources without having to deal with high-latency.

It has been demonstrated in previous section that computational offloading can decrease energy consumption and increase performance if the offloaded task requires a lot of computational processing (4, 7, 8, 9, 14). The offloading can happen either to a remote server or a device in proximity (mobile devices, cloudlets etc.). As connecting to the cloud involves higher latency than connecting to nearby devices, it should be more feasible to use these resources instead.

Current mobile games are already with PC-like features and with the emerging of D2D mobile games, there is a need to balance the computational load for the devices. When dealing with mobile devices in proximity, everyone has limited battery life. As different offloading tasks are given to a device, instead of gaining energy they spend it. This raises the question, whether users are willing to share their already limited processing capabilities with other devices as it increases energy consumption.

As a result, D2D offloading has been proposed. Instead of processing a task for another device, computational offloading can also be used to share the intermediate results of a processed task. By doing this, the device is alleviated from processing a task that another device has already processed. To validate our ideas, we built a D2D framework and a simple 2D game.

3.1 Summary

In order to counter the problems of high-latency when trying to acquire computational resources we propose to use nearby devices. The devices can establish connection between themselves using Bluetooth. By taking into account the next generation D2D mobile games, it may be highly beneficial to be able to share data with nearby devices. This could lead to smoother gameplay, lower loading times, better battery consumption and better overall user satisfaction.

4

D2D Mobile Gaming

In the previous section, we talked about the possibility of sharing the computational load between multiple devices in proximity. When dealing with games, more specifically multiplayer games, it is possible to reduce load times and power consumption by sharing the computational tasks between nearby devices. For example when two devices are playing a multiplayer game, it is unnecessary for both of them to load each image, sprite, model etc. locally. Instead they can divide these tasks between them and share the results with each other.

4.1 D2D framework

In order for devices to offload computational heavy tasks between nearby devices, we created a D2D framework. The general schema for D2D code offloading can be seen on Figure 4.1. When comparing it to the general schema in Figure 2.1 on page 4, it can be seen that instead of a cloud server, a collection of nearby devices are used for offloading purposes.

Figure 4.1 shows the model for code offloading in D2D framework. The application(1), is installed in both the device(2) and nearby devices(4). When the bar method is called, the offloading framework decides which nearby device to use as a slave and sends the necessary data(3) to it. The slave then executes the method(5) using Java reflection and sends the result(6) back to the application in the master device.

4.2 Implementation

The framework is implemented for Android devices. Java reflection is used to offload to other devices. Bluetooth is used to establish a connection between devices in proximity. This is

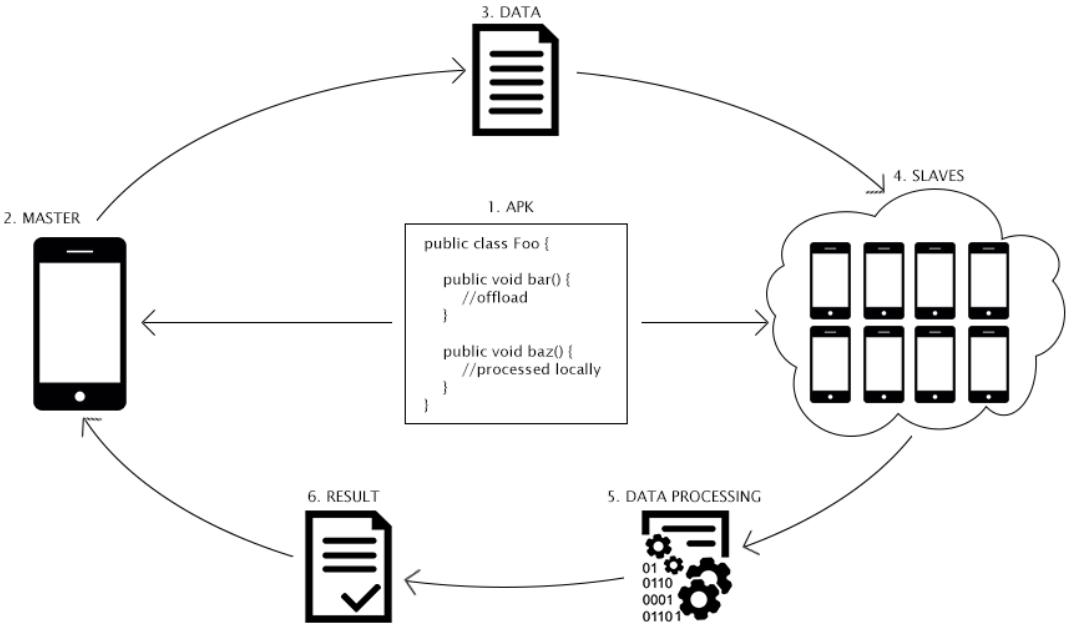

Figure 4.1: D2D code offloading schema

done by creating an insecure radio frequency communication(RFCOMM) *BluetoothSocket* between the devices. Once the runtime execution details of the code are captured, they can be sent back and forth in the communication using *ObjectInputStream* and *ObjectOutputStream*, respectively. Capturing the runtime details of the code allows the devices to reconstruct the code in environments that share the same execution properties.

Figure 4.2 shows the generalized architecture of D2D framework. Each of these devices has an application(*.apk file*) and the framework installed. The *Connection Manager* establishes a connection between the devices. The *System profiler* is in charge of collecting data about the device, application and network. The *Code profiler* determines what code to offload based on annotations added by the developer of the application. Both devices(1 and 2) have a *role*, which can be either master or slave. The role is assigned by the frameworks *Orchestrator*. Once the slave is chosen, the master sends a request to get the intermediate results from the slave. The slave handles the request and sends the result back to the master. This sequence can be seen in Figure 4.3.

If there are multiple slaves to select from, greedy algorithm is used to make the decision. This algorithm selects the best choice available at the current time without taking into account

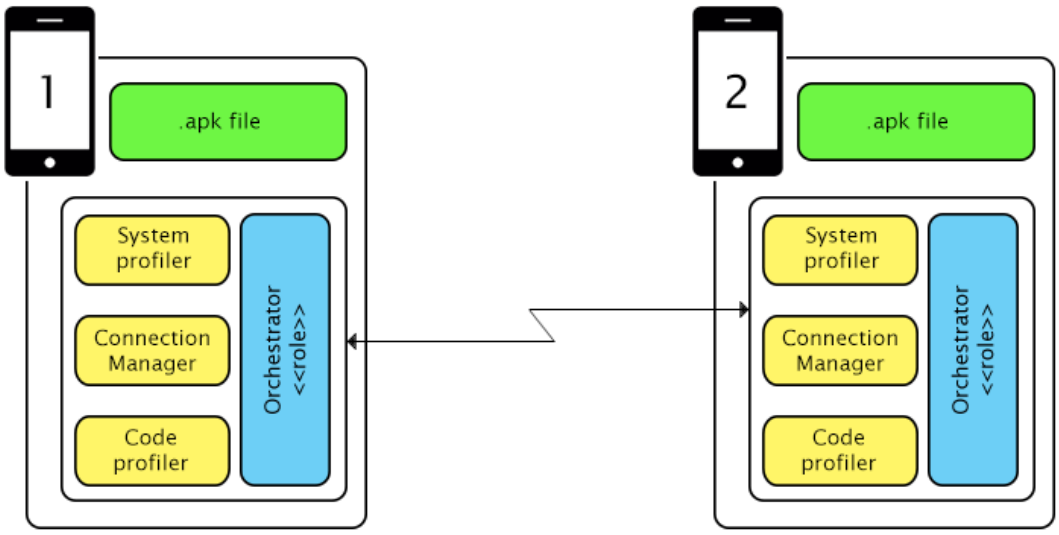

Figure 4.2: Generalized architecture of D2D Framework

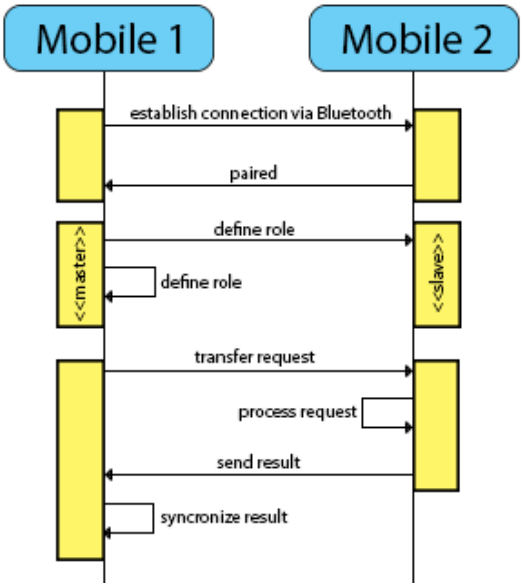

Figure 4.3: Sequence diagram of D2D framework

possible future consequences [15].

The method shown on Snippet 2 returns the connected devices descendingly ordered by CPU idleness and that are currently not busy. The *deviceList* contains all the devices MAC

[15] http://www.encyclopediaofmath.org/index.php/Greedy_algorithm

Snippet 2 Greedy algorithm in D2D framework

```
public List<String> greedyDecision(ConnectedDeviceList deviceList) {
    List <String> devices = new ArrayList<String>();
    statustable.sortDescStatus("cpuIdleness");
    for (int i=0; i<deviceList.size(); i++) {
        if (deviceList.getJobStatusList().get(i) == true) {
            devices.add(deviceList.getDeviceList().get(i));
        }
    }

    return devices;
}
```

addresses and the *statustable* has the data collected for each device that is currently connected to the master device. The device is busy if it is in process of offloading data. First devices' MAC address returned from this method will be assigned to be in the slave role.

The framework also includes a custom logger, that uses the devices database to store information about the offloading process. By being able to download the contents of the database table into the device, it is possible to analyse the data to improve the framework.

4.3 Validation

To validate the framework, we built a battle game. The game was implemented using the Android 2D OpenGL Game Engine called AndEngine [16]. GLES2 version of the AndEngine was used for the game, which is based on OpenGL ES 2.0 [17]. PhysicsBox2DExtension [18] was used to create the physics of the world.

The game consists of two levels populated with enemies, that need to be destroyed. Level 1 of the game can be seen in Figure 4.4 and level 2 from Figure 4.5. The user has control over the character(*wizard*) in the middle of the screen. The *knight* and the *ghost* serve the purpose of enemies. The available controls are as follows: moving to the sides, jumping and shooting projectiles. Projectiles are shot by triggering a touch event in the desired direction.

[16] https://github.com/nicolasgramlich/AndEngine
[17] http://www.andengine.org/blog/2012/06/andengine-gles2-old-and-new-news/
[18] https://github.com/nicolasgramlich/AndEnginePhysicsBox2DExtension

Figure 4.4: Level 1 of the battle game

A total of 22 images were used for the sprites in the game. These images include the tiles, animated characters, backgrounds and controls for the game. The art used in the game comes from PlatForge [19]. It is estimated that 46 sprites and 35 bodies of these sprites are created during one gameplay. For example in Figure 4.4 there are a total of 16 sprites visible and Figure 4.5 displays 15 sprites. If the player dies, the scoreboard is shown, which can be seen in Figure 4.6. This screen consists of two sprites, the background image and a back button. These do not include the projectiles, as these are created when an attack is initiated and destroyed after contact or reaching the end destination.

Bodies are used in order to add physics attributes like weight, elasticity, fixture, movement etc. to sprites. They are divided into three types: static, kinematic and dynamic. As the name says, static bodies are static, they will not move(e.g. tiles, buttons etc.). Kinematic and dynamic bodies are used when movement is necessary. Kinematic bodies do not interact with the forces(e.g. gravity) of the physics world, instead they can be given a velocity at which they move in a certain direction. In contrast dynamic bodies can be fully simulated and they interact with other body types. The movement of dynamic bodies is created by adding a force to them

[19]https://play.google.com/store/apps/details?id=edu.elon.honors.price.maker

in a specific direction. For instance the *wizard* is a dynamic body type, the player has control over the forces that manipulate the body by using movement commands.

Figure 4.5: Level 2 of the battle game

It was decided that the offloading shall be tested on loading the sprites of the game as this could in theory greatly decrease the loading times. The method to be offloaded was hard-coded into the application. This means that every time the game is run, the framework will try to offload the loading of sprites to other nearby devices at runtime. Figure 3 shows one part of the code that is offloaded. This code is responsible for creating the *ITextureRegion* for the mountain that is accessed by the game, once the loading of level 1 is initiated.

Snippet 3 Code that loads the mountain image used for background in level 1 of the game

```
ITextureRegion mountain =
    BitmapTextureAtlasTextureRegionFactory.createFromAsset(
        backgroundTextureAtlas,
        activity,
        "mountain.png"
    );
```

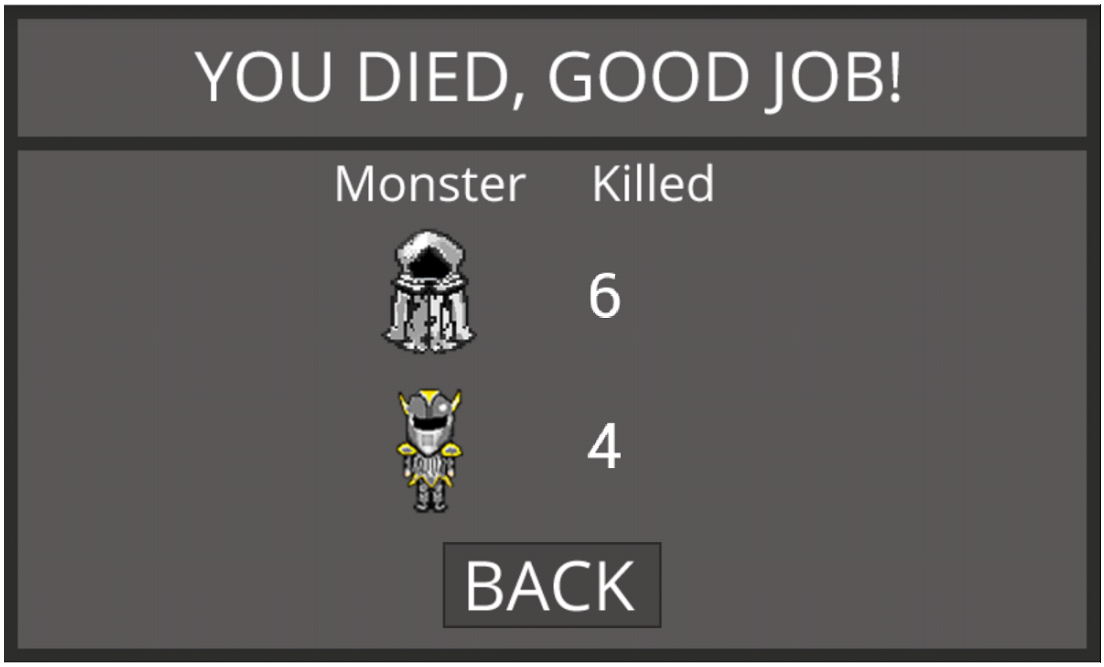

Figure 4.6: Scoreboard of the game, once the player dies

The source code for the game can be obtained from GitHub [20] The game requires the device to have touch screen capabilities [21] and atleast Android Ice Cream Sandwich (4.0) platform. However the required minimal platform for the device is Android Jelly Bean (4.1) as this is the requirement of the framework itself.

For the validation we used a Samsung Galaxy S3 I9300 equipped with Android Jelly Bean (4.1.2) and a Sony Xperia Z1 that has Android KitKat (4.4.4). The setup can be seen in Figure 4.7 and the results can be seen in Figure 4.8. PowerTutor (20) was used for the measurement of energy. The first column shows the energy usage when not using D2D framework and the second column shows when it was used.

When playing the game normally, the master device used 3252 J and the slave 4321 J of energy. However when they shared results, the master device spent only 457 J and the slave used 4786 J of energy. It can be seen that by using the framework, the slave device had to spend a little more energy than usual, but the master device was able to save almost six times the energy used when compared to normal usage. Combined the devices ended up saving energy.

[20] https://github.com/huberflores/CaseStudy-QoS-CodeOffloading
[21] https://source.android.com/devices/input/touch-devices.html

Figure 4.7: Setup of the devices used for measuring the power consumption

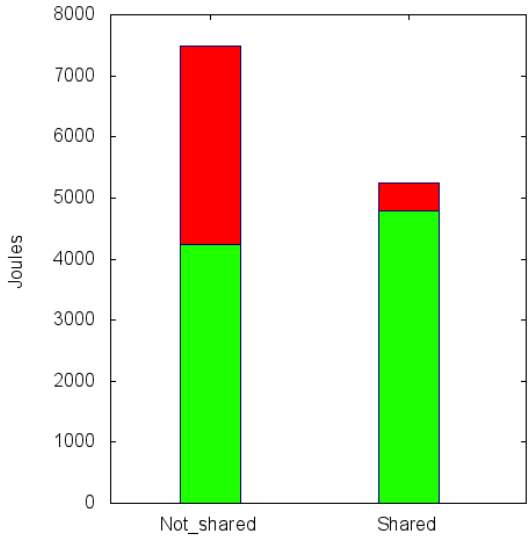

Figure 4.8: Diagram showing the power consumption of two devices- without using the framework and with using it

4.4 Summary

Developing the game and making a use case out of it, made it possible to demonstrate the abilities of the framework. The master device succeeded in getting the results for a task, that had already been done by the slave and ended up saving energy. Although the slave ends up wasting more energy then it would normally, the two devices combined used less energy in total.

5

Conclusions

Smartphones keep evolving as consumers demand for better performance and battery life. For example, performance can be boosted by equipping the device with a more powerful processor and more RAM. But this in consequence usually increases the battery usage. It has been suggested to use computational offloading to augment the devices capabilities as it has been shown that by using computational offloading techniques, it is possible to increase both the performance and battery life of smartphones.

There are different solutions proposed for code offloading which include using cloudlets, VMs located in the cloud etc. However it is also possible to harvest the resources of nearby devices as discussed earlier. This can be highly advantageous when dealing with D2D mobile games.

By giving tasks to other nearby smartphones to solve means that the offloading process comes at the expense of other devices' battery life. If only one device does the offloading for others then it results in being disadvantageous for it and highly rewarding for others. It is the job of the framework to determine how to offload, so that all devices can benefit from it.

However, it is also possible to share the intermediate results of tasks with other nearby devices. This enables to alleviate the device of processing said task. We developed a prototype and results give positive insights about the applicability of the technique.

5. CONCLUSIONS

6

Future Research Directions

As the main target of D2D mobile games are nearby devices, computational offloading techniques should be used to balance the processing of computationally heavy tasks. However there are still some drawbacks to computational offloading.

For instance, current frameworks designed for offloading include poor profilers. ThinkAir suggests gathering data about the offloading processes to improve this. However with crowdsourcing as proposed by EMCO it is possible to take it a step further. By gathering data about different offloading processes and environments to a central cloud database, it enables profilers to more accurately determine the best solution for offloading.

Because of constant changes to applications, network infrastructures and devices, we believe that by creating a hybrid framework designed to offload would be most suitable. By allowing different types of connections to be established(3G, 4G, WiFi, Bluetooth etc.) and targets(cloud, cloudlet, mobile devices etc.) to choose for offloading, it is possible to accommodate different needs for both the user and the application. Cloud servers might grant access to more computational power as opposed to nearby devices, but this comes at a cost of using connections other than Bluetooth, therefore requires more energy and may suffer from high-latency.

On the other hand in 2014, Google announced a new Android runtime(ART). The main purpose of this is to replace Dalvik, the VM on which Android Java code is executed on [22]. ART is designed to be compatible with Dalvik Executable format and Dex bytecode specification, however some techniques that work on Dalvik do not work on ART [23].

[22] http://anandtech.com/show/8231/a-closer-look-at-android-runtime-art-in-android-l/

[23] https://source.android.com/devices/tech/dalvik/

6. FUTURE RESEARCH DIRECTIONS

The biggest change coming with ART is that it implements Ahead-of-Time(AOT) compilation instead of Just-in-Time(JIT) as it was with Dalvik [22]. This means that the application is compiled once during the first execution and every subsequent executions will not compile it again, instead reuse the already compiled native code. Optimizing and compiling the entirety of code only once results in decreasing overall power consumption. Because of this, the first-run of an application takes considerably more time than in the case of Dalvik. However the tests indicate a performance boost of roughly two times compared to Dalvik [22].

In theory, this does not however have a negative effect to the D2D framework, as the already compiled code is still reusable by another device by sharing the necessary results before the first run of the application on another device.

7

Sisukokkuvõte

Tänu läbimurretele mobiilsete seadmete ja sotsiaalvõrgustikes vastastikuse mobiilse suhtlemise valdkondades on seadmelt-seadmele (ingl. k. device-to-device) mobiilsed mängud muutunud aktuaalseks trendiks. Selleks, et säästa rakenduste poolt nõutavat energiat ja kiirendada nende reaktsiooniaega, on võimalikuks vahendiks kasutada koodi mahalaadimist pilve vahendusel või seadmelt-seadmele. Teatavasti on andmevahetuses eelistatud madal latentsusaeg, mille tõttu on seadmelt-seadmele mahalaadimine sobilikum. Sellegipoolest ei ole lähedal asuvale seadmele mahalaadimine praktikas otstarbekas, sest kasutaja ei pruugi olla nõus teise seadme poolt edastatud ülesande lahendamises, kuna sellega kaasneb lisanduv energia kadu. Antud töös läheneme probleemile uuest küljest: selle asemel, et lasta teisel seadmel töö ära teha on võimalik kasutada juba lahendatud ülesannete tulemusi. Püstitatud eesmärgi saavutamiseks arendati välja raamistik ja teostati juhtumiuuring. Valideerimise tulemusele põhinedes leidsime, et lähedal asuvate, omavahel ühendatud seadmete puhul on võimalik vähendada rakenduse koormust.

Võtmesõnad:
Android, Koodi mahalaadimine, Bluetooth

8

Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, Mihkel Visnapuu (date of birth: 14/08/1992), herewith grant the University of Tartu a free permit (non-exclusive licence) to:

1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

1.2. make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

Device-to-Device Mobile Gaming
supervised by Huber Flores and Satish Narayana Srirama,

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, 14.05.2015

8. LICENCE

Bibliography

[1] H. Flores, P. Hui, S. Tarkoma, Y. Li, S. Srirama, R. Buyya, Mobile code offloading: from concept to practice and beyond, Communications Magazine, IEEE 53 (3) (2015) 80–88. 1, 8, 9

[2] H. Flores, S. N. Srirama, C. Paniagua, A generic middleware framework for handling process intensive hybrid cloud services from mobiles, in: Proceedings of the 9th International Conference on Advances in Mobile Computing and Multimedia, ACM, 2011, pp. 87–94. 3

[3] H. Flores, S. N. Srirama, Mobile cloud middleware, Journal of Systems and Software 92 (2014) 82–94. 3

[4] K. Kumar, Y.-H. Lu, Cloud computing for mobile users: Can offloading computation save energy?, Computer (4) (2010) 51–56. 3, 4, 11

[5] H. Flores, S. Srirama, Mobile code offloading: should it be a local decision or global inference?, in: Proceeding of the 11th annual international conference on Mobile systems, applications, and services (MobiSys 2013), ACM, pp. 539–540. 3, 8

[6] M. Satyanarayanan, P. Bahl, R. Caceres, N. Davies, The case for vm-based cloudlets in mobile computing, Pervasive Computing, IEEE 8 (4) (2009) 14–23. 3, 6

[7] P. Bahl, R. Y. Han, L. E. Li, M. Satyanarayanan, Advancing the state of mobile cloud computing, in: Proceedings of the third ACM workshop on Mobile cloud computing and services, ACM, 2012, pp. 21–28. 3, 6, 7, 11

[8] S. Kosta, A. Aucinas, P. Hui, R. Mortier, X. Zhang, Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading, in: INFOCOM, 2012 Proceedings IEEE, IEEE, 2012, pp. 945–953. 3, 7, 11

[9] M. S. Gordon, D. A. Jamshidi, S. Mahlke, Z. M. Mao, X. Chen, Comet: Code offload by migrating execution transparently, in: Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation, USENIX, 2012, pp. 93–106. 3, 7, 11

[10] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, A. Patti, Clonecloud: elastic execution between mobile device and cloud, in: Proceedings of the sixth conference on Computer systems, ACM, 2011, pp. 301–314. 3

[11] C. Shi, V. Lakafosis, M. H. Ammar, E. W. Zegura, Serendipity: enabling remote computing among intermittently connected mobile devices, in: Proceedings of the thirteenth ACM international symposium on Mobile Ad Hoc Networking and Computing, ACM, 2012, pp. 145–154. 3, 9

[12] M. Satyanarayanan, Pervasive computing: Vision and challenges, Personal Communications, IEEE 8 (4) (2001) 10–17. 6

[13] R. Balan, J. Flinn, M. Satyanarayanan, S. Sinnamohideen, H.-I. Yang, The case for cyber foraging, in: Proceedings of the 10th workshop on ACM SIGOPS European workshop, ACM, 2002, pp. 87–92. 6

[14] H. Flores, S. Srirama, Adaptive code offloading for mobile cloud applications: Exploiting fuzzy sets and evidence-based learning, in: Proceeding of the fourth ACM workshop on Mobile cloud computing and services, ACM, 2013, pp. 9–16. 8, 11

[15] N. Fernando, S. W. Loke, W. Rahayu, Mobile cloud computing: A survey, Future Generation Computer Systems 29 (1) (2013) 84–106. 8, 9

[16] D. Jian-jun, X. Heng-cheng, A distributed online test system based on bluetooth technology, in: Software Engineering (WCSE), 2010 Second World Congress on, Vol. 1, IEEE, 2010, pp. 15–17. 8

[17] J.-H. Chang, L. Tassiulas, Energy conserving routing in wireless ad-hoc networks, in: INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, Vol. 1, IEEE, 2000, pp. 22–31. 9

[18] H. Qi, A. Gani, Research on mobile cloud computing: Review, trend and perspectives, in: Digital Information and Communication Technology and it's Applications (DICTAP), 2012 Second International Conference on, ieee, 2012, pp. 195–202. 9

[19] E. E. Marinelli, Hyrax: cloud computing on mobile devices using mapreduce, Tech. rep., DTIC Document (2009). 9

[20] Z. Yang, Powertutor-a power monitor for android-based mobile platforms, EECS, University of Michigan, retrieved September 2. 19