Teemu Kämäräinen

Design, Implementation and Evaluation of a Distributed Mobile Cloud Gaming System

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 26.5.2014

Thesis supervisor:

Prof. Antti Ylä-jääski

Thesis advisor:

D.Sc. (Tech.) Matti Siekkinen

Aalto University
School of Electrical
Engineering

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE
MASTER'S THESIS

Author: Teemu Kämäräinen

Title: Design, Implementation and Evaluation of a Distributed Mobile Cloud Gaming System

Date: 26.5.2014 Language: English Number of pages:8+56

Department of Computer Science and Engineering

Professorship: Data Communications Software Code: T-110

Supervisor: Prof. Antti Ylä-jääski

Advisor: D.Sc. (Tech.) Matti Siekkinen

Cloud gaming where the games are rendered on distant cloud servers and streamed to thin clients is currently gaining ground. It enables relatively weak computational devices such as mobile phones to be used to play games that normally couldn't be run on the devices. Cloud gaming is very susceptible to latency though and previous research has shown that the current distant cloud infrastructure is not adequate especially for the most demanding games.

This thesis proposes a more distributed cloud gaming infrastructure focusing on the use of cloudlets on the network edges. An open-source Cloudlet Remote Gaming Platform prototype is built using the cloudlet model and the response delay in different mobile networks and cloud server locations is measured. The study concludes that the use of cloudlets on network edges could benefit the QoE for the users especially when playing the most demanding fast-paced games. The power consumption of the mobile device in cloud gaming is also measured using different networks showing that by switching to Wi-Fi connection and by using external displays and game controllers the power consumption of the mobile device could be decreased on top of improving the QoE for the user.

Keywords: Cloud gaming, latency, cloudlet, LTE, Wi-fi

AALTO-YLIOPISTO
SÄHKOTEKNIIKAN KORKEAKOULU

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä: Teemu Kämäräinen

Työn nimi: Hajautetun pilvipelaamisen mobiililaitteilla mahdollistavan järjestelmän suunnittelu, toteutus ja arviointi

Päivämäärä: 26.5.2014 Kieli: Englanti Sivumäärä:8+56

Tietotekniikan laitos

Professuuri: Tietoliikenneohjelmistot Koodi: T-110

Valvoja: Prof. Antti Ylä-jääski

Ohjaaja: TkT Matti Siekkinen

Pilvipelaamisella tarkoitetaan pelien etäpelaamista kevyillä päätelaitteilla. Pelit suoritetaan pilvipalvelimilla ja videokuva lähetetään päätelaitteisiin verkon yli. Tämän ansiosta laitteita, joiden suorituskyky ei normaalisti riittäisi pelin suorittamiseen voidaan käyttää pelien pelaamiseen. Pilvipelaaminen on erittäin herkkä verkkoviiveille ja aikaisempi tutkimus on osoittanut nykyisen pilvi-infrastruktuurin riittämättömäksi kaikkein vaativimmille peleille.

Tässä työssä esitetään hajautettu pilvipelaamisen malli keskittyen erityisesti cloudlettien eli pienten käyttäjän lähellä olevien pilvien käyttöön verkon laidoilla. Työtä varten luotiin cloudlet-mallia hyödyntävä prototyyppi etäpelaamista varten ja mitattiin käyttäjän kokema viive eri mobiiliverkoissa ja pilvipalvelimen sijainneissa. Mittaukset osoittivat, että cloudlettien avulla voidaan myös kaikkein vaativimpia pelejä suorittaa etänä verkon yli. Työssä mitattiin myös mobiililaitteiden virrankulutus pilvipelaamisen aikana. Erityisesti keskityttiin tutkimaan voisiko mobiililaitteen virrankulutusta vähentää käyttämällä ulkoista näyttöä pelaamiseen. Virtamittaukset osoittivat, että Wi-fi-verkkoon kytkettynä ja ulkoista näyttöä käyttäen voidaan pilvipelaamisen virrankulutusta vähentää huomattavasti mobiiliilaitteilla.

Avainsanat: Cloud gaming, latency, cloudlet, LTE, Wi-fi

Preface

Haluan kiittää Professori Antti Ylä-Jääskeä ja ohjaajaani Matti Siekkistä hyvästä ohjauksesta ja ennakkoluulottomasta asenteesta aihevalintaani kohtaan. Lisäksi haluan kiittää Yu Xiaoa, Miika Komua, Taneli Hukkista ja Olli Kiljusta yhteistyöstä Cloudlet-aiheen parissa.

Erityiskiitos myös perheelleni jatkuvasta tuesta diplomityön ja koko opiskelujeni aikana.

Otaniemi, 26.5.2014

Teemu V. E. Kämäräinen

Contents

Symbols and abbreviations

Abbreviations

API	Application programming interface
CDN	Content delivery network
CPU	Central processing unit
DRX	Discontinuous Reception
EC2	Amazon Elastic Compute Cloud
FPS	First-person shooter
GA	GamingAnywhere
GaaS	Gaming as a Service
GPU	Graphics processing unit
HTML5	Hypertext Markup Language (fifth revision)
IP	Internet protocol
ISP	Internet Service Provider
JSON	JavaScript Object Notation
LTE	Long Term Evolution
MCC	Mobile Cloud Computing
MCG	Mobile Cloud Gaming
ND	Network delay
OD	Playout delay
OS	Operating system
PD	Processing delay
PC	Personal computer
QEMU	Quick EMUlator
QoE	Quality of Experience
QoS	Quality of Service
RD	Response delay
RPG	Roleplaying game
RTT	Round-trip time
TCP	Transmission Control Protocol
URL	Uniform Resource Identifier
VGA	Video Graphics Array
VM	Virtual Machine
WAN	Wide area network
WP8	Windows Phone 8

1 Introduction

Mobile devices such as tablets and smart phones have lots of conflicting requirements. They should at the same time be lighter and smaller but more computationally powerful and energy-efficient. Although the development of mobile devices over the past decade has been rapid, so has the development of non-mobile devices. This shows that non-mobile devices such as desktop computers can always outperform small hand-held devices. One way of bridging this gap is to perform heavy tasks remotely on another instance.

Widely deployed data centers have enabled countless applications for mobile devices to utilize cloud computing [31]. Applications in the field of file sharing and video streaming are the most popular. Data centers can also provide computational offloading for the mobile device. Applications usually offload computing intensive background tasks to the cloud. This is possible because these tasks normally don't have stringent latency requirements.

Cloud gaming or Games-On-Demand is a technology which offloads the tasks of graphic rendering, computation and storage into clouds away from the end-user device [27]. This is sometimes called GaaS or Gaming as a Service [34]. It is a new kind of service, which combines the successful concepts of Cloud Computing and Online Gaming [23]. Cloud gaming enables relatively weak thin-clients to be used for playing games which normally couldn't be natively played on the device. Moving game rendering to the cloud is not trivial though. Cloud games are very susceptible to latency and cannot rely on the same delay compensation techniques as normal online games do. In addition, not all games are equally friendly to cloud gaming. Some require less strict latency requirements than others [27]. The overall delay perceived by the user in cloud gaming is called the response delay. It is defined as the time it takes for a user's input to be processed on the mobile device, transferred through the network, a corresponding video frame to be processed on the server, sent back to client and displayed on the mobile device.

Moving gaming to the cloud increases the requirements of network quality of service (QoS). Cloud gaming could possibly have the most stringent demands on network QoS of any other cloud service. It requires both high down-link bandwidth and low latency. Cloud gaming providers such as OnLive[1] have had to operate on numerous data centers across the US alone just to support users in the same country. Although cloud gaming puts tremendous challenges for the network it also opens new business opportunities in a growing market, which have been recognized by the game industry [20]. Cloud gaming industry is still relatively young and the question of how far can the remote rendering happen is still under research. Massive distant cloud servers such as the Amazon EC2 instances are an appealing choice but their long distance from the end-user might inflict too high latency to make the business-model feasible everywhere. Another more distributed network model might be needed.

A cloudlet is a computer or a cluster of computers connected at the edge of the Internet to provide low-latency access to computing resources for mobile devices [30].

[1] OnLive: http://www.onlive.com/

Cloudlets have been visioned to liberate mobile devices from resource constraints and help overcome the latency issues of distant data centers. Speech recognition, natural language processing, computer vision and graphics, machine learning, augmented reality and other compute-intensive applications could leverage this new technology. [30] Remote gaming could possibly also benefit from cloudlets. Gaming requires lots of computing power and most games require low latency to achieve a good user experience. Cloudlets are by definition connected to a mobile device by a low-latency one-hop connection. Cloudlet computing lets many users share a host platform of one or more multi-core Central processing units (CPUs) and Graphics processing units (GPUs). The cloudlet can be as small as a single computer with one multi-core CPU and GPU or as big as a data-center with thousands of servers. It has been expected that cloudlets could bring back the old one-host-multiuser paradigm. [28]

Using only cloudlets for cloud gaming could be difficult though because the availability of cloudlets could vary depending on the location of the user. In addition it may not be necessary to always use cloudlets for cloud gaming because of the varying latency requirements of different games. A distributed cloud infrastructure using cloudlets when necessary and a distant cloud when cloudlets are not available or not necessary might improve the quality of experience (QoE) for the user and widen the user base available for a cloud gaming service provider.

Latency and bandwidth issues are not the only ones hindering the use of mobile devices as a platform for PC or console games. For instance these games are usually meant to be played with external controls using a large display. The next section will in addition to the delay problem, list the other existing issues in using mobile devices for cloud gaming by defining the problem statement of this thesis.

1.1 Problem statement

Modern desktop PCs outperform mobile devices and often laptops as well both in CPU and GPU performance. Hardware requirements together with a wider range of platforms make mobile devices simply incapable of running the most recent video games. Cloud gaming enables these modern games to be played on virtually any device capable of decoding a high-quality video stream and forwarding user's control input back to the remote server. Latency issues from the client device to the cloud server can however lower the experienced quality of service (QoS) beyond acceptable levels when using distant cloud servers. Using cloudlets for bringing the cloud closer to the user could potentially solve these issues. On the other hand the availability of nearby cloudlets and the difference in latency requirements in different games could have an impact on which location the cloud game should be initiated.

Latency and other network requirements are not the only obstacles on using mobile devices for playing desktop PC games. PC games are usually designed for a much larger screen than what is available on most mobile devices. Controls of such games are also often designed only for desktop computers. Some games and game types support gamepads but often desktop computer games are meant to be played with a mouse and keyboard combination. This issue has to be addressed also when considering playing desktop games on mobile devices. Furthermore the power

constraints of the mobile phone might limit longer gaming sessions as the constant streaming of the game might drain out the battery of the device quite fast.

To summarize six issues have been found to be addressed with mobile cloud gaming:

1. The strict network QoS requirements of remote gaming might be too much for distant cloud servers to be used as the remote server. This should specially be the case in the most latency-sensitive games.

2. The possibly low availability of nearby resources and the different requirements of games need the cloud infrastructure to be dispersed, allowing the game to be rendered as close as possible taking into account the requirements of the game instance and the availability of resources.

3. The screen size of the mobile device might have a big impact on QoE as well because desktop computer games are designed for a larger screen.

4. The controls of desktop computer games are also often optimized for gamepads or for the mouse and keyboard combination, which might be problematic because mobile devices usually only possess a touch screen.

5. Mobile devices and other suitable thin clients have a variety of popular platforms meaning a separate native client would have to be implemented for each platform.

6. Finally the power consumed when streaming video and forwarding user controls might limit the feasibility of using mobile devices for long gaming sessions.

Section 1.2 discusses how this thesis plans to solve the problems found in this section.

1.2 Objectives of the thesis

This thesis outlines a design for a distributed cloud gaming infrastructure to support the use of both cloudlets and distant cloud servers such as the Amazon EC2. The new needed components are described and the benefits gained from the new cloud server locations enabled by the new system design are assessed.

The focus is on assessing the feasibility of using cloudlets for remote gaming. The evaluation is done by building a complete open-source Cloudlet Remote Gaming Platform from existing open source components modifying them when necessary. The main focus is on assessing the overall response delay perceived by the user in different cloud server locations. The comparisons are made by running a reference game streaming server on a distant cloud server and comparing its performance with the Cloudlet Gaming Platform in different networks and cloud server locations.

Problem 3 concerning the screen size of the mobile device depicted in section 1.1 could be avoided partly by letting the user display the game on an available external monitor, TV or even a public display letting the mobile device only act

as a controller. The implementation is done by dividing the display and control logic of the remote gaming software and designing a way for the display and mobile device to discover each other. The new system design allows the mobile device not only to use nearby computing resources such as cloudlets but also other resources such as external displays or game controllers. The case where the game has no gamepad support is also considered by constructing a way to simulate the mouse and keyboard combination using an external controller such as the gamepad.

The platform issue of Problem 5 is partly related to the public display scenario. Public displays often run everything in a browser, which is also a common platform between various mobile operating systems (iOS, Android, WP8 etc.). For this issue it must be tested if the display part and possibly also the control part of the game application could be streamed completely using a browser without any plug-ins or external applications. An HTML5 alternative is constructed for the software used in the Cloudlet Remote Gaming Platform.

Finally the Problem 6 is assessed by measuring the power consumption of the device in different networks. It is also tested if substantial power consumption benefits could be gained by using a public display or an external controller with the mobile device when playing a game.

1.3 Structure and focus

The rest of the thesis is structured as follows. Chapter 2 covers briefly mobile cloud computing before presenting the concept of cloudlets and the used software for cloudlet initiation. Chapter 3 covers the necessary background and previous research in the fields of cloud and remote gaming. It also discusses the special role of latency in cloud gaming.

Chapters 4 and 5 cover in detail the overall design of the distributed cloud gaming infrastructure and the constructed Cloudlet Remote Gaming Platform both with the native remote gaming software and the HTML5 alternative. The platform is evaluated against an instance running in a distant cloud server in Chapter 6 where the power consumption of the mobile device in different scenarios is also measured. Finally the thesis concludes with discussion and conclusions in Chapters 7 and 8 respectively.

The main focus is on assessing the benefits gained from the new distributed cloud gaming design. This is done by constructing the needed Cloudlet Remote Gaming Platform and assessing its performance in different scenarios. A design of the whole system is given although the implementation of the resource management between the cloud components is left for further research.

2 Cloudlets and Cloud Computing

Chapters 2 and 3 introduce the background concepts needed in the implementation of the distributed cloud gaming system. This section briefly summarizes the concept of Mobile Cloud Computing (MCC) and defines the concept of cloudlets. The open-source software implementation of the cloudlet-model called Elijah is also presented and the needed property of VGA passthrough in the GPU-powered virtual machines is discussed.

2.1 Mobile cloud computing

Cloud computing is a model for enabling access to a shared pool of configurable computing resources [29]. In the model computational resources are not physically present in the device itself. Instead the resources are remotely accessed through an Internet connection. [26]

Cloud computing has become a very popular environment for hosting web applications and services. Data centers can utilize resources more effectively resulting in lower costs and the ability to increase capacity whenever needed. [15] Cloud computing can also potentially save energy for mobile users. Additional benefits include rapid launch of new services and a more reliable infrastructure in the case of hardware failures. [26]

Mobile Cloud Computing (MCC) extends this model for mobile devices such as mobile phones and tablets. Cloud offloading is an important aspect in mobile computing. It can be defined as the availability of cloud computing services in a mobile ecosystem [17] or briefly by the availability of cloud data processing and storage for mobile users [18].

The offloading of resources from mobile and other devices to a more resource-rich environment is sometimes called cyber foraging. The concept is not new dating all the way back to mid-1990s. Mobile devices have less computing power than their server counterparts. [19] The two main categories in mobile devices are mobile phones and tablets. They both interact with the user in a similar fashion, usually with the aid of a touch screen. Tablets usually have a slightly larger display. They both access the cloud usually through a web browser or a thin client application. [17]

Offloading mobile applications to the cloud should in theory save battery life of the mobile device as the computation is outsourced to the cloud. There is however a rise in the network usage when using the cloud which drains additional power from the device. [33] Kumar et al. have concluded that energy can be saved in cloud offloading if a task requires high amount of computation and a low amount of network traffic [26]. However energy savings are not the only benefits of cloud offloading. In some cases the computations couldn't be handled in the mobile device at all which leaves offloading as an only option. [33]

To be able to utilize remote cloud servers, mobile devices need a network connection with low enough latency and high enough bandwidth. Long Term Evolution (LTE) networks, often abbreviated 4G, enable high-speed data access for mobile

devices. LTE networks are targeted to provide an increased peak data rate of 100 Mbps and an improvement for latency thanks to a simplified IP-based design [6]. An LTE and a 3G network as well as a Wi-Fi connection is used in the measurements.

Large data centers or groups of data centers such as the Amazon EC2 are an obvious choice for cyber foraging. It has been argued though if these kinds of remote data centers have the capabilities to support the most latency-sensitive applications. Cloud providers such as Amazon usually offer their resources in a relatively small number of data centers in locations often chosen to minimize costs rather than the latency [15]. Users expect desktop-like experience from latency sensitive applications such as cloud gaming. Between the mobile device and the distant cloud server, the data traverses through many wired and wireless links, and network elements which all add to the overall response delay perceived by the user. [33] To bridge this gap in the mobile off-loading scene - a new element called the cloudlet has been visioned to be placed between the cloud and the mobile device.

2.2 Cloudlets

Running resource-heavy applications in the cloud is a good way of offloading computation from the mobile device. The wide area network (WAN) latencies can however degrade the user experience of low-latency applications. In the new cloudlet-model the mobile device would still only act as a thin client. However the computation would be offloaded into a nearby cloudlet instead of a distant cloud server. [30]

Mobile hardware has always less computing power than dedicated client or server hardware. This limitation occurs because of strict requirements for such attributes as battery life, size and weight. [30] Distant data centers can provide virtually unlimited storage and computational power for tasks without strict latency requirements. Real-time applications can however have extremely strict latency requirements to achieve a sufficient user experience. Application fields such as speech recognition, natural language processing, computer vision and graphics, machine learning, augmented reality, planning and decision-making are examples of fields, which require such low latencies [30].

A cloudlet is a trusted, resource-rich computer or cluster of computers that's well-connected to the Internet and available for use by nearby mobile devices. It is a new architectural element in a 3-tier hierarchy: mobile device, cloudlet, cloud [19]. The architecture is presented in Figure 1. The cloudlet and the mobile device are within one-hop proximity to minimize the latency. Cloudlets are planned to be very self-managing with minimal upkeep needed. The main differences between clouds and cloudlets concern their state, management, environment, ownership, network and number of users. Cloudlets have only soft state, meaning they won't store any data after the user's session. If something needs to be preserved, it must be stored to the cloud. Cloudlets are also designed to be self-managing and decentralized. They only serve a few users at a time although with an extremely low latency. Cloudlets are planned to be very easily deployable by essentially being a small scaled data center. They are visioned to be deployed in a similar fashion as Wi-Fi access points are available today. [30]

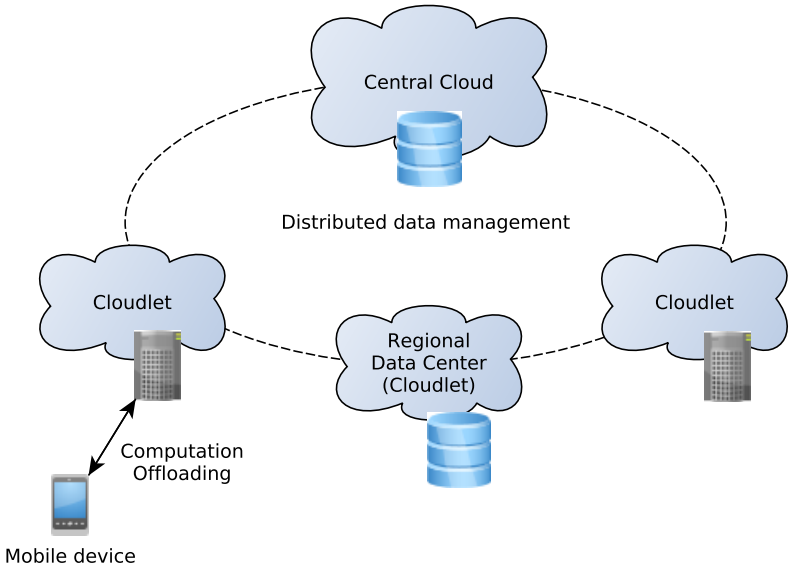

Figure 1: The cloudlet architecture.

Cloudlets utilize rapidly deployed Virtual Machines (VMs) which the client can customize freely. The customization, shown in Figure 2, is done prior to deployment by modifying a base disk image, which consists of a stock installation of a popular operating system such as Windows or Linux. This customization is packed in a VM Overlay, which is calculated as the difference between the base image and the customized image of the user. The customized image has the application and all necessary libraries installed to run the desired software. The overlay can be stored in the client device and uploaded to the cloudlet before launching the VM. In the proposed cloudlet gaming system it is more feasible for separate distributed data storage to hold the actual overlays as the sizes of games and thus the overlays could be measured in gigabytes. This may lead to some loss in the ubiquity of the system, as the distributed data storage must have the VM overlay before a mobile device can use it.

High computing power and low-latency connection of the cloudlet make it possi-

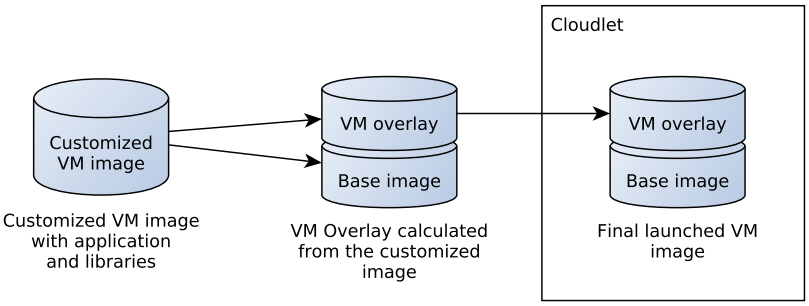

Figure 2: The dynamic synthesis of the VM image.

ble to re-introduce a one-host-multiuser computing paradigm in the form of cloudlet-screen computing [28]. In this model all of the computation are done in the cloudlet and the resulting image output is streamed to the client. In addition the cloudlet receives control commands from the client. In this paradigm, the cloudlet also needs the power of a GPU on top of the CPU to be able to serve the client effectively. The GPU is used to render the graphics on the host side and can be also used to encode the resulting video into a lossy video stream to be further streamed for the client. This paradigm is utilized in the prototype implementation presented in Chapter 5.

Cloudlets have been previously envisioned to be used together with the concept of public displays. In the work of Clinch et al. cloudlets are used together with public displays for display appropriation. Clinch et al. focused on the question how close does the cloudlet need to be relative to the client. They conclude that the closer the better but that for minimally immersive/interactive applications, cloudlets can leverage commercial cloud infrastructure. This translates to this study as it is shown that the most demanding games require cloudlets for a decent QoE.

2.3 Elijah / QEMU

Elijah is a software implementation of the Cloudlet-model developed in the Carnegie-Mellon University [4]. It is capable of rapidly deploying a VM image and also has the needed tools to create a base image as well as an overlay image.

Elijah utilizes a so-called VM synthesis where VMs are generated dynamically from an overlay, which is the difference of a base VM image and the modified custom VM image needed by the client application. The steps of the synthesis are presented in Figure 3. The base VMs are preloaded into the cloudlet. First the mobile device discovers the available cloudlet and negotiates the use of it. After negotiation the mobile device informs the cloudlet about the overlay wanted for the VM. The cloudlet fetches the overlay and applies it to the base VM image. When the VM is ready, the cloudlet informs the mobile device of this and the mobile device can start using the launch VM. In the original model the VM overlay is sent by the mobile device itself. In the new model the mobile device only identifies the overlay wanted and the cloudlet is responsible of retrieving the overlay from a predefined location or from a location specified by the mobile device.

Elijah has several optimizations to speed up the synthesis such as deduplication, pipelining of the processes and a process called early start. Elijah is used in the Cloudlet Remote Gaming prototype assessed in this work. The changes applied to it are described in Chapter 5.

Elijah uses software called QEMU to launch the Virtual Machines. QEMU is an open-source emulator and virtualization software which is able to run a complete operating system as just another task [8]. It supports a range of processors and operating systems. QEMU is currently also capable of VGA passthrough, which is needed for running GPU-powered applications effectively on virtual machines.

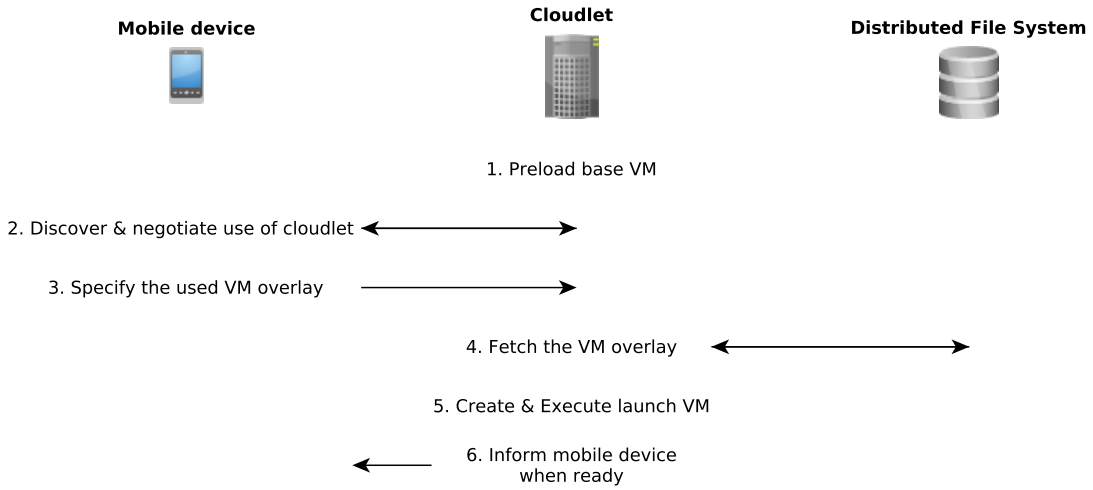

Figure 3: The VM synthesis timeline. Derived from [30] and [19].

2.4 GPU utilization

Graphics processing unit (GPU) is a special hardware needed to rapidly accelerate the creation of graphics. It is a necessary hardware to be able to run PC games and other graphically intensive applications. Virtual machines typically use an emulated graphics adapter to display the desktop or render simple graphics. The emulated adapter uses the host's GPU through a software interface. The performance of these emulated devices are currently not good enough for high-end gaming. VGA Passthrough overcomes this issue by assigning a physical GPU card for the Virtual Machine. The VM has direct control of the GPU and can utilize the same drivers as if the card was running on the host. This way the VM can handle GPU-intensive applications such as games with very little overhead compared to the host machine.

VGA passthrough is utilized in the prototype of the Cloudlet Remote Gaming System by assigning each virtual machine with its own GPU. The drawback of this approach is that each running VM needs a dedicated GPU card on the host machine. The most effective way of utilizing the GPUs would be to share their computational power between the VMs on the fly. However this technology currently only exists in dedicated closed-source data center software.

GPU passthrough is used in commercial large-scale virtualization solutions. Nvidia's GRID [2] software uses dedicated software and hardware to passthrough GPUs to virtual machines. It uses graphic cards with multiple GPUs. For example a server with two Nvidia's GRID K1 graphic cards has a total number of 8 GPUs capable of running 8 fully accelerated virtual machines. In addition each GPU can be virtually divided for up to 8 users. This shows the scalability of the GPU powered VMs. The implementation of the Cloudlet Remote Gaming Platform in Chapter 5 expands this concept also for PCs with virtually any configuration of graphics cards without the need of expensive dedicated hardware.

[2] http://www.nvidia.com/object/nvidia-grid.html

3 Cloud Gaming

This chapter introduces the concepts of remote and cloud gaming, discusses the role of delay in cloud gaming and introduces the remote gaming software used in the implementation of the Cloudlet Remote Gaming Platform.

3.1 Cloud gaming and remote gaming

Based on what kind of tasks are handled by the cloud, or in other words the utilization of cloud resources, cloud-based gaming can be classified into different categories. So far the most common strategy is to use the cloud only for content distribution and file synchronization between different devices. Game distributors can effectively use cloud storage for sending the game software for the client's personal device.

Another approach is to run online multiplayer servers in the cloud and handle only the state changes of connected clients. In this approach the game itself is still rendered on the client side. The highest level of cloud utilization for gaming is to render the gaming scenes entirely on the cloud server side. In the context of this work cloud gaming is defined as the remote rendering of the game application on the cloud server. The client only plays back the remote video stream sent by the server and sends user's control input back to the cloud server.

Figure 4 presents the framework of a typical cloud gaming platform. Two sep-

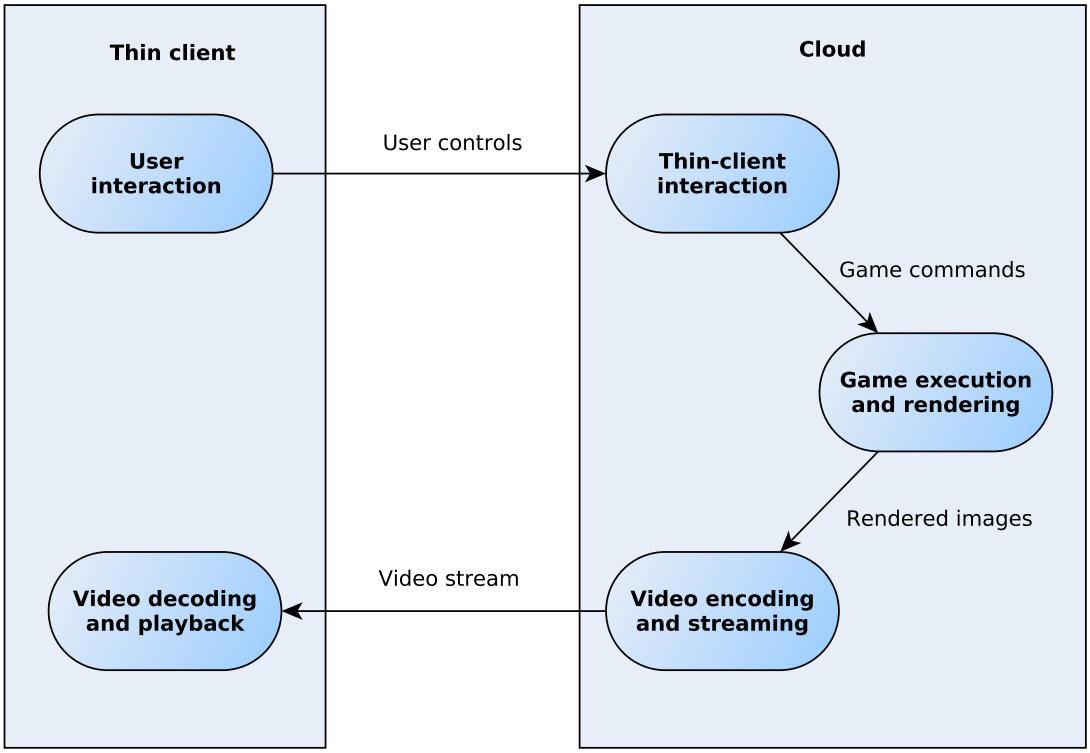

Figure 4: Framework of a cloud gaming platform. Adapted from [31].

arate flows of information can be found in the framework: the video flow and the control flow. The game is entirely executed and rendered on the cloud platform. The rendered images are captured and encoded into a more compact form often with lossy encoders. The cloud platform is also responsible of streaming the resulting video stream to the client. The client decodes the encoded video stream and plays it back on the client device.

The second flow of information is the control flow. The thin client is responsible of capturing user's interactions, packing them into standardized format and sending them to the cloud server. The cloud server receives the control messages and processes them on the host machine as if they had occurred locally.

Widespread deployment of cloud gaming has been predicted to become a reality in the upcoming years thanks to advances in cloud technology [31]. A few companies such as Gaikai (purchased by Sony), OnLive and G-cluster already deploy a variety of solutions for cloud gaming. [12] It has been expected that the cloud gaming market could expand massively in the upcoming years: nine times over the period of 2011 to 2017 reaching 8 billion US dollars [3].

The main advantages of moving games to the cloud are the far less strict requirements for the client hardware and the possibility to remove almost all architectural limitations [14]. Potentially millions of new devices such as tablets and mobile phones could gain access to games, which would run normally only on high-end desktop PCs. The mobile devices could simply act as thin clients without the need to install complete game engines on the devices. As the game data is stored in the cloud, the loss of data is less likely to incur and can also be moved across different devices and networks to achieve a seamless gaming experience [12].

Thanks to cloud gaming, users also wouldn't have to upgrade their devices frequently to support the latest games anymore and possibly would also enable the users to play more games thanks to the lowered hardware and software costs [21]. Cloud gaming has also the potential to reduce the power consumption of the mobile device as most of the computation is occurring at the cloud server [34]. According to Huan et al. [21] cloud gaming could also reduce production costs and increase net revenue for the developers by not having to develop for several platforms at the same time.

In addition to cloud gaming, remote gaming solutions where the game is rendered on another instance in the same local network, are also gaining ground. Nvidia has launched a game streaming product which can stream games from a PC to a special mobile device called Nvidia Shield[3]. The game streaming currently works only in the same local network using Wi-Fi access. Another such solution is called Steam In-home Streaming[4], which is currently in beta phase. It can stream games from one PC to another also only in the same local network. It remains to be seen if these solutions will be expanded to work through the public Internet infrastructure in the future.

[3]http://shield.nvidia.com/

[4]http://steamcommunity.com/groups/homestream

3.2 Mobile Cloud Gaming

The mobile gaming market is rapidly expanding [12]. Although mobile devices are becoming more powerful all the time, they still have inherent restrictions such as limited battery lifetime and lower computational capacity than for example desktop PCs or game consoles. Cloud-based mobile computing could possibly remove these restrictions. Leung and Chen define [31] Mobile Cloud Gaming (MCG) as interactive gaming utilizing mobile devices that access the cloud as an external resource for processing the game scenarios and interactions, and to enable advanced features such as cross-platform operations, battery conservation, and computational capacity improvement.

In cloud gaming systems the game applications are run on a powerful server. The games are streamed to thin-clients, which can have relatively low computational power. The thin-client can be any network connected device capable of receiving the user's inputs, sending them to the cloud and receiving and playing the video stream. The most demanding part for the thin client is the rapid decoding of the high-resolution video stream. Luckily many devices such as phones and tablets already possess hardware-based video decoders capable of decoding the video stream on the fly.

Gaming applications are very popular also on mobile devices. A significant portion of all applications sold on different application markets consists of games. Additionally gaming has been predicted to move more and more towards mobile devices away from the traditional consoles. Cloud gaming is not yet massively popular on mobile platforms. Onlive[5] is currently the only cloud gaming provider with a mobile client. Overall the possible user base does exist also for Mobile Cloud Gaming.

The delivery paradigm used in cloud gaming is sometimes also called Gaming as a Service (GaaS) [34]. Soliman et al. [34] observe four main concerns limiting the widespread deployment of GaaS: User responsiveness, video quality, service quality and operating cost. Further limitations include low battery life, small screen size and limited control options compared to PCs or game consoles. The special role of response delay in cloud gaming is discussed more in the next chapter. Other limitations when implementing the Cloudlet Remote Gaming Platform are also considered further in Chapter 5.

3.3 Role of response delay in Cloud gaming

Cloud gaming forces strict requirements for both the cloud server and the underlying network. The cloud needs to be able to render high-performance 3D-graphics and encode the graphics into a high-resolution video stream. This has to be done in the matter of milliseconds since the overall response delay with the user has to be kept as low as possible. [31] The time difference between a user's command input and the corresponding in-game action appearing on the screen can be defined as the response delay (RD) [21].

[5] http://www.onlive.com/

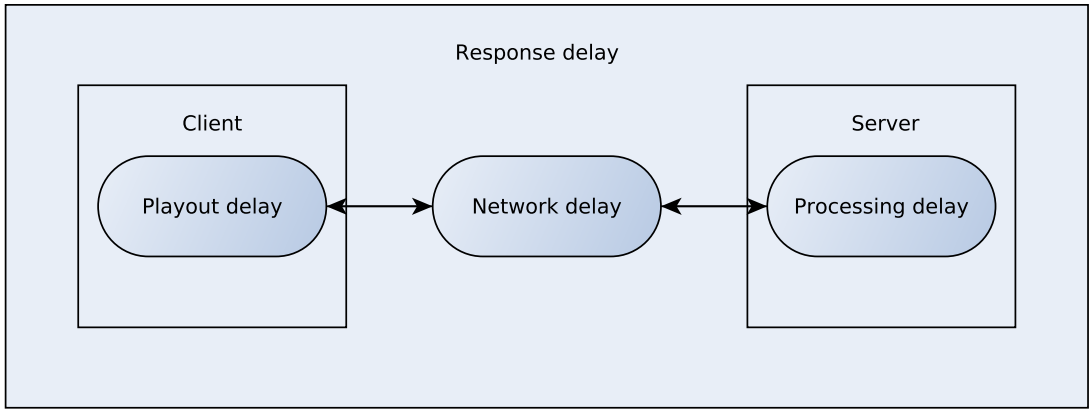

Figure 5: The three main components of the overall response delay.

The overall response delay (RD) can be divided into three components: processing delay (PD), network delay (ND) and playout delay (OD) [14]. The processing delay sums up all the delays at the server side between receiving a command from the user and submitting the corresponding video frame back to the user. Network delay is the round-trip time (RTT) between the thin client and the server and means the time it takes for a command to traverse through the network and a video frame to come back to the client. Playout delay is the time it takes for the client to display a frame to the user after receiving it from the server. The delay components are presented in Figure 5.

The playout delay and the processing delay can be improved by using faster video codecs or for example hardware acceleration to handle the encoding and decoding of the video frames. The network latency is somewhat a harder problem. One of the most critical challenges of cloud gaming is the strict latency requirement for the connection between the client and the cloud. Cloud data center locations are usually decided based on cooling opportunities and electricity costs, rather than minimal latency to end-users [15]. In addition the network latency is always bounded at least by the speed of light.

Previous studies have shown that the response delay for the most demanding games must be at least below 100 ms [20] [32]. In cloud gaming this includes processing the user's interaction, rendering the new state, encoding and compressing the output video stream as well as the network delay both to and from the server (round-trip-time). In regular gaming the delay is only an issue in multiplayer gaming where as in cloud gaming the delay is also present in single player games. In addition traditional multiplayer games are often able to hide the latency to some extent by rendering the game locally and notifying the server of the changes afterwards. This is no longer possible in cloud gaming because the rendering is moved entirely to the cloud [32].

Jarchel et al. show [25] that the current data center scheme of the Amazon EC2 is not optimal for meeting the strict latency requirements of on-demand gaming. The study concludes that either many more large data centers or a large number of

specialized "smart-edge" servers should be added to meet the latency requirements of the most demanding games. This thesis further contributes to this proposal by assessing the possibility of using cloudlets as these smart-edge servers. The results are also assessed by comparing them to thresholds found for different game types in previous research.

3.4 Latency requirements of different game types

Latency determines how players experience online gameplay quality of experience and also has a direct correlation to user's performance in the game. Depending on the type of game and its interactions, even a 100 ms delay can cause up to 35 % decrease in player's performance [16]. The work in [25] suggests that 20 ms should be subtracted from this to take into account the encoding and decoding delays although they concur that the estimate is quite optimistic and the delay of the process could be larger.

Different games and game types have varying latency requirements in cloud gaming [27]. This means the overall response delay can be higher for some games without lowering the quality of experience (QoE) perceived by the player. This might have implications to the optimal location of the cloud server if several alternatives exist. Lee et al. have proposed a model [27] which can predict the susceptibility of a game to latency in terms of its QoE in cloud gaming. The key essence in their model is that games with more frequent screen changes and control inputs suffer more from higher response delay. They show that first-person-shooter (FPS) games have the strictest delay requirements. Roleplaying games (RPGs) and action games vary in their requirements depending on the nature of the particular game. Fast-paced RPGs could be more susceptible to latency than slower paced action games and vice versa.

The classical categorization of game types in QoE studies has been adopted widely from a study by Claypool and Claypool [16]. They divide games in three categories: Omnipresent (e.g. real-time strategy games), Third-Person Avatar (e.g role-playing games) and First Person Avatar (e.g. First Person Shooters). They give thresholds for latencies that shouldn't be exceeded for the performance of the player not to decrease. The limits are 100 ms for First Person Avatar games, 500 ms for Third Person Avatar games and 1000 ms for Omnipresent games. These limits, the 100 ms rule for FPS games in particular, have been widely used in the literature as thresholds for acceptable levels of delay. However these limits were obtained by measuring the performance of players under different delays, not the perceived QoE. The results were also obtained using online games not cloud games. Online game servers and clients can compensate for latency as the rendering of the game is still happening on the client side. As cloud game servers cannot do this compensation, it can be claimed that these limits are quite high for acceptable delays for cloud gaming.

Jarschel et al. have researched the perceived QoE of users in different network conditions specifically in cloud gaming [23]. They concluded that packet loss and latency are the key components in defining the perceived QoE. Packet loss affects

Table 1: Delay thresholds acquired for different game types.

Game type	Delay range (ms) (MOS 4 to 3)	Threshold chosen
Fast-paced	40-80	60
Medium-paced	80-180	130
Slow-paced	80-300	190

the experience by leaving out control inputs or by worsening the image quality. In fast-paced games the delay component becomes the dominant metric affecting the QoE. The delay ranges where the perceived QoE decreases below acceptable levels can be determined by interpolating the measurement results by Jarchel et al. Mean Opinion Score (MOS) value of 4 is used as a point where the user starts to notice the delay and the MOS value of 3 where the delay already begins to hinder the gameplay. Table 1 shows the delay ranges where the QoE crucially drops (MOS drops from 4 to 3) and the midpoints which were chosen to be thresholds used in the delay measurements.

In the measurements the focus is on the delay because the system is designed to bring the most demanding cloud games closer to the user by using a distributed cloud gaming infrastructure. The limits obtained from the study by Jarschel et al. are used to evaluate the measurement results.

3.5 Remote Gaming Software

The use of traditional remote desktop software has been previously proposed to be used also in cloud gaming. A measurement study [13] has however shown that these solutions don't provide high enough frames per second (FPS) for cloud gaming. Dedicated cloud gaming software has emerged since to enable a low enough response delay for cloud gaming. Commercial remote streaming and cloud gaming software include for example Onlive, StreamMyGame[6], Splashtop[7] and Kainy[8]. They all encode the game video into a video stream and transport it between the server and the client.

Most available cloud gaming systems are closed-source proprietary software. This limits their assessment for scientific purposes. They also cannot be modified or extended to test new ideas. GamingAnywhere (GA) [20] is an open-source cloud gaming software developed in the National Taiwan Ocean University. It is intended to be used by researchers and engineers for testing new cloud gaming systems and ideas. Thanks to the open-source approach the system can be easily extended and reconfigured.

GA's main design objectives have been extensibility, portability, configurability and openness. It consists of client software running on a separate thin client and

[6]http://streammygame.com/
[7]http://www.splashtop.com/personal
[8]http://www.kainy.com/

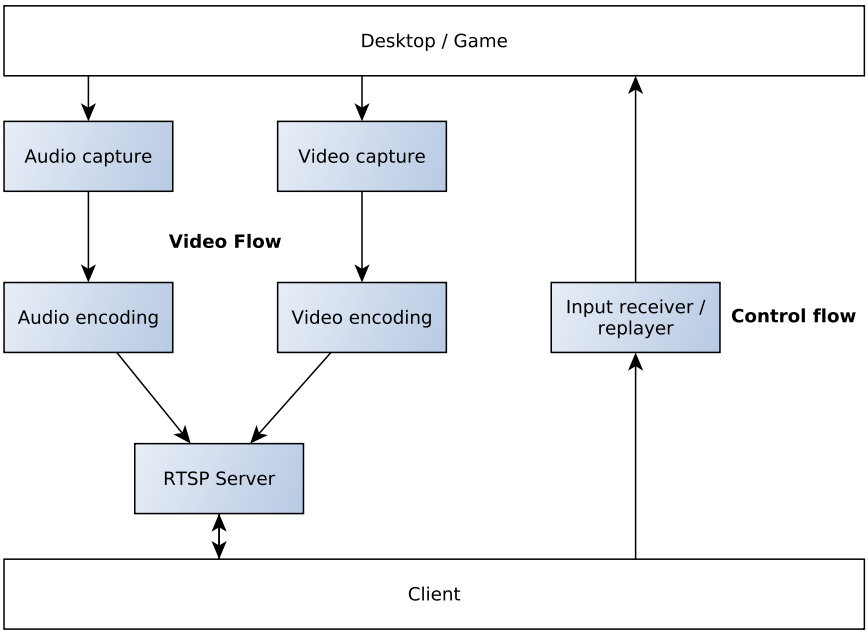

Figure 6: The server modules of the GamingAnywhere software. Adapted from [21].

a server running on another instance. The server has six distinct modules that enable games to be played over a network. The input handling module runs on a thread which receives input messages from the client and replays them on the host machine. The audio and video capture modules capture game screens and audio from the running game. The captured multimedia is piped into encoder modules that compress the streams for delivery. A Real Time Streaming Protocol (RTSP) module is responsible for launching the encoders and handling the data flows between the server and the client. The relationships between the server modules are presented in Figure 6.

The client part of the program also has a separate input handling module that captures events made by game players and sends them to the server. Client's RTSP module receives the video and audio stream from the server after which the frames are passed to a frame buffering and decoding module that outputs the game scenes on the screen of the client device. The client modules are presented in Figure 7.

The GA server currently works on Windows, Linux and OS X with a client available also for the mobile Android operating system. GamingAnywhere is used to set up the testbed for cloudlet gaming. The platform and modifications are explained further in Chapter 5. The GA server built for Linux and the Android client are used in the measurements.

The tasks performed by the various modules are responsible for the overall response delay experienced by the user. The processing delay components of the server can be further divided into four main components: memory copy, format conversion, video encoding and packetization. Memory copy means capturing the raw image from the game or desktop, format conversion is the conversion of color-space, video

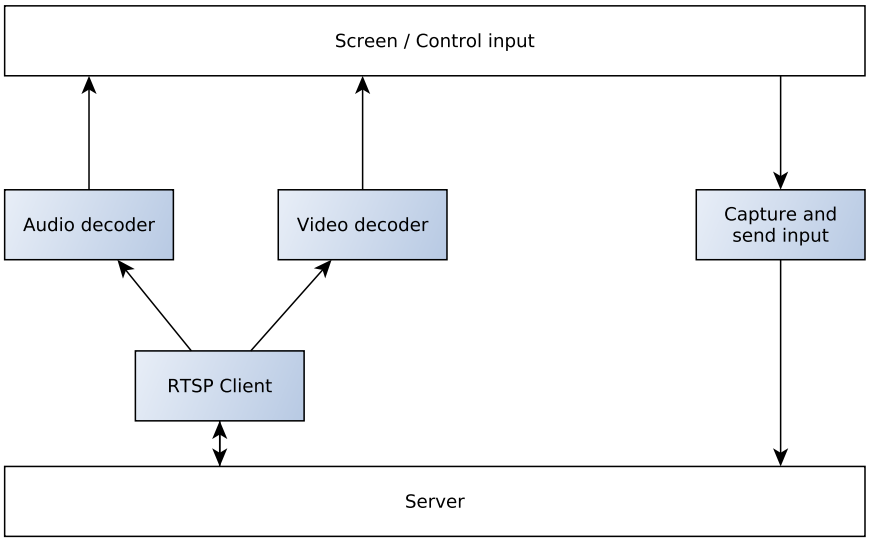

Figure 7: The client modules of the GamingAnywhere software. Adapted from [21].

encoding means the compression of the stream and packetization is the segmentation of the data to be sent. On the client side the delay can be broken down into three components: frame buffering, video decoding and screen rendering. Frame buffering means receiving the necessary packets for one video frame, video decoding means decompressing the encoded video stream and screen rendering means displaying the actual image on the device. [21] The components of the overall response delay are presented in Figure 8. These are the delays caused by the game streaming software without the network delay.

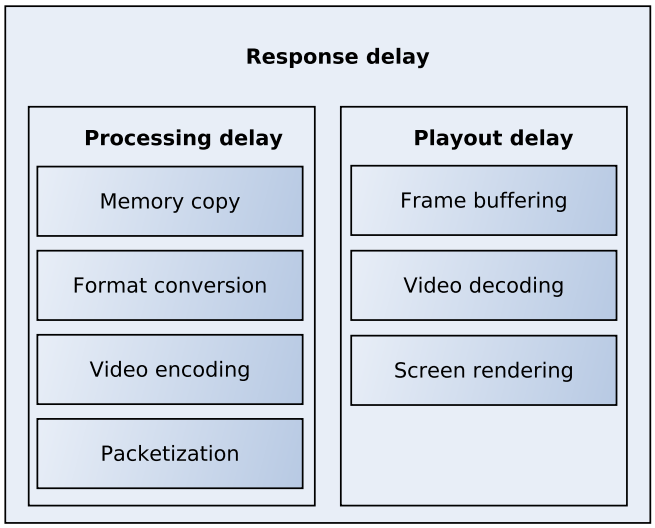

Figure 8: The breakdown of the overall response delay.

4 Distributed Cloud Gaming System

This chapter describes the overall design of the new distributed cloud gaming system. It describes the tasks of a new element called the Resource Provisioning Server and presents the role of public displays and smart TVs in the new design. Finally the workflow of instantiating a virtual machine in the new proposed system is listed.

4.1 Overview

High network latency between the mobile device and cloud infrastructure is a key challenge in cloud gaming. This thesis presents a decentralized solution to deploy the cloud gaming servers on hosts closer to the mobile device to reduce latency. The proposed model includes the public cloud like the Amazon EC2. In addition an option is added to instantiate the cloud server also on an enterprise cloud which could be located for instance in operator premises. The proposed model also supports private clouds that could be for example a single PC connected to a Wi-Fi access point at home. Chapter 5 presents a prototype Cloudlet Remote Gaming Platform that could be deployed anywhere between the user and the public cloud. The new distributed cloud infrastructure should be able to better meet the requirements of different game types and prevailing network conditions in cloud gaming. The new infrastructure is presented in Figure 9.

Game servers are run in virtual machines (VMs) in all deployment locations of the system. The computing environment of a game server is isolated from that of others, while a game server can serve one to multiple game clients, depending on the design of the game. In this thesis, the focus is on the scenarios where the server renders the game graphics and the captured A/V frames on the game server are encoded and streamed to game clients. In these scenarios, except enough CPU cores and memory/disk space, criteria for selecting host machines also includes: the availability of GPUs and the access from VMs to the GPUs, and the network connectivity that should provide enough bandwidth while offering low latency for data

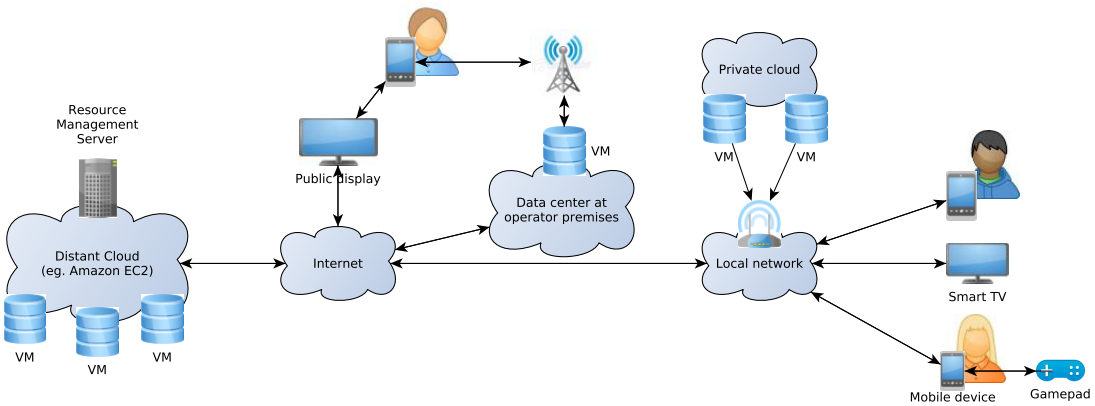

Figure 9: The main components of the proposed distributed system.

.

communications between the game server and its clients. In practice, a centralized Resource Provisioning Server takes care of the deployment of game servers, including the selection of host machines, the initiation of the VMs, and the configuration of the game servers in different computing and networking environments. This thesis leaves out the final implementation of the provisioning service for future work and concentrates on assessing the gained benefits of switching the location of the cloud gaming server by building a prototype Cloudlet Remote Gaming Platform and deploying it in various locations. The provisioning service can be later implemented based on the findings of this thesis.

Game clients are responsible for decoding the video streams coming from the game server, and forwarding user input from input consoles to the game server. Game clients run on network terminals that are connected to the Internet through Wi-Fi or cellular networks. These network terminals are equipped with or connected to input and/or output consoles. Examples of these network terminals include smartphones, tablets, smart TVs, laptops and set-top boxes for televisions. In this thesis the focus is on using mobile devices for remote gaming so the test scenarios will be implemented using wireless cellular and Wi-Fi connections using smart phones and tablets.

Depending on the input/output consoles available, users can choose to play games, for example, directly on smartphones using the touch screen of the terminal or by using a gamepad connected to the mobile device as an input console while showing the video on a TV or public display. In the second case, the input and output consoles are connected to two different network terminals, the smartphone and the set-top box of the TV or public display. There will be one game client running on each network terminal, while both game clients are connected to the same game server running in the cloud. When starting a game, the configuration of the game server will be adapted to that of game clients. This approach will potentially save energy compared to the alternative of directly connecting a monitor to a mobile device.

4.2 Resource Provisioning Server

The Resource Provisioning Server is a web service that handles requests from the gaming clients. The requests indicate the device identities of the input/output consoles and the identity of the game. The provisioning service is responsible for deploying the corresponding gaming server on a proper host machine, taking the quality of user experience into account.

Gaming servers run in virtual machines (VMs). The VM synthesis techniques developed by Ha et al. [19] for VM provisioning are applied, and their open source library called elijah-provisioning[9] is used for constructing base VMs and VM overlays and for implementing the dynamic VM synthesis.

A base image is typically a stock installation of a popular OS such as Windows or Linux. The VM in which the gaming server is running is often called the launch

[9]https://github.com/cmusatyalab/elijah-provisioning

VM. It is created by installing relevant software into a base VM. A VM overlay is the binary difference between the launch VM and the base VM. It is usually compressed. Two overlays are usually created, one from the VM disk snapshot and the other from the VM memory snapshot.

Given the VM overlays and the base VMs from which the overlays are derived, the launch VM can be instantiated. The procedure can be seen as a reversed process of overlay generation. In practice, it can be assumed that base images containing popular operating systems are widely available in the distributed cloud infrastructure. For each game, its overlays can be stored in a portal server or on a distributed file system that can be searched. When the game is ordered, the overlays can be delivered to the host machine where the gaming server is scheduled to run.

When a client request arrives, the provisioning service first searches for the base images and overlays, and then checks the requirements for creating the launch VM, including the processing capability of CPU/GPU, the size of memory/disk and the network connectivity. After that, it selects a host machine that can provide enough resources and is as close as possible to the game client taking into account the delay requirements of the particular game. If there is a host machine available in the same network with the game clients, this host machine will be selected. Otherwise, the host machine that is located with more network hops away will be considered. When the client is done using the cloudlet, the resource is freed and the provisioning server is informed. The provisioning server keeps track of the available cloudlets.

In practice, a proprietary network may manage the public displays, and the host machines may belong to different owners, which make it difficult to discover and query the resources. In the presented design the public displays can be either centrally managed by the Resource Provisioning Server or be completely independent and discovered by the mobile device.

4.3 Public displays and smart TVs

One of the issues listed in the problem statement of this thesis is that the small screen size of mobile devices might affect the user experience when playing games designed to be played on the PC or a game console. For this reason the cloudlet gaming system should be able to use external displays such as smart TVs or public displays to view the game. One option would be to connect a display cable between the mobile phone and the external display or stream the game image from the mobile device even further into the smart TV or public display. In the designed system this is not recommended because the smart TVs and public displays are connected to the Internet and can stream the game image themselves, saving crucial battery life of the mobile device and avoiding additional delays.

It has been visioned that soon public displays would be widely deployed which could be also publicly interacted with in a step towards pervasive and ubiquitous computing [11]. Such a display would be located in a public space such as a bar or an airport and the users would have limited access on displaying content on the display. Various prototypes have been already designed and implemented. The levels of manipulating user-generated content on the displays vary. The most common way

of manipulating the content on the public display is through a web browser for which the user simply inputs an URL. So to be able to play a game using a public display, the game view would have to be able to run inside a browser. Another option would be to have the remote gaming application already installed on the machine connected to the public display. In this scenario the mobile device and public display could exchange information through the logic of the particular application. The web browser option would be more universal and could also have other advantages and use cases as well.

Smart TVs are also rapidly gaining popularity. Smart TVs are televisions with network connectivity and an operating system capable of running various applications and streaming media from the Internet or from the user's home network. [1] For the purpose of remote gaming smart TVs can be incorporated into the distributed cloud gaming system in two ways. A native application can be designed for each smart TV platform solely for the purpose of remote gaming. Another option is to use a web application since smart TVs usually have at least a web browser application preinstalled. In Chapter 5 an HTML5 based web application is designed which could potentially be used in cloud gaming on public displays and smart TVs without the need to install third party applications on the devices.

4.4 Service discovery

In order for the proposed system to function properly, the different devices and servers need to be able to discover each other's presence. The Resource Provisioning Server described earlier is responsible of keeping track of free cloud and cloudlet servers capable of launching the VMs. The Provisioning Server is a simple web application with a database of cloudlets and other possible cloud locations. It can accept new cloudlet entries either from a separate web service or directly from the mobile device. As the system is designed to be as universal as possible, it's important for the provisioning service to accept cloudlets from different providers. In the current setup the mobile device can discover cloudlets in the same network using Universal Plug and Play (UPnP) discovery. So the mobile device can inform the Provisioning Server of nearby cloudlets on top of getting possible cloud server locations from the provisioning server's database.

Public displays can be discovered in a multitude of ways in the system depending on the implementation of the particular display infrastructure. Presuming the remote gaming software can be preinstalled to the system running the public displays, then the mobile device can simply read an identifying Quick Response Code (QR) of the display and inform the provisioning service about this display when requesting a cloud gaming server. The provisioning server can contact the preinstalled remote gaming application on the public display system, which can in turn instantiate the video stream with the cloud gaming server. This is not however possible if the public displays are run by a different instance and cannot have the remote gaming software preinstalled. In this case the game video is received using a web application.

This approach presumes that there is a possibility for the user to open a URL address in the public display using for instance a touch screen input. The user enters

the URL of the remote gaming service, which opens a web socket connection to the server. QR codes can again be used to identify the particular public display. The web page displays a QR code that can be read in the mobile application to identify where to stream the game video.

Finally external game controllers can be connected to the mobile device in the regular manner using a USB cable or wireless technologies such as Bluetooth or Wi-Fi. The remote gaming software is responsible of giving the user an option to choose the desired controller and transmitting the input commands to the cloud gaming server.

4.5 Workflow of Initializing a Game

A good cloud gaming system should 1) allow a player to easily set up a game from his/her mobile device, and to choose his/her favorite input/output consoles from the ones available nearby, 2) provide high-quality user experience, and 3) make the procedure of game installation and initialization as transparent as possible for the player. The proposed design presented is designed to satisfy the above criteria.

The workflow between the mobile device and the distributed gaming system when launching an instance is describes as below.

1) The game client initiates the connection by contacting the Resource Provisioning Server. The client sends information about cloudlets discovered in the local network, the identity of the selected game and user login information if necessary.

2) The Resource Provisioning Server running in the cloud checks the deployment requirements of the selected game, including the CPU, GPU, memory and disk size, finds the most suiting available host, and ships the images of the gaming server

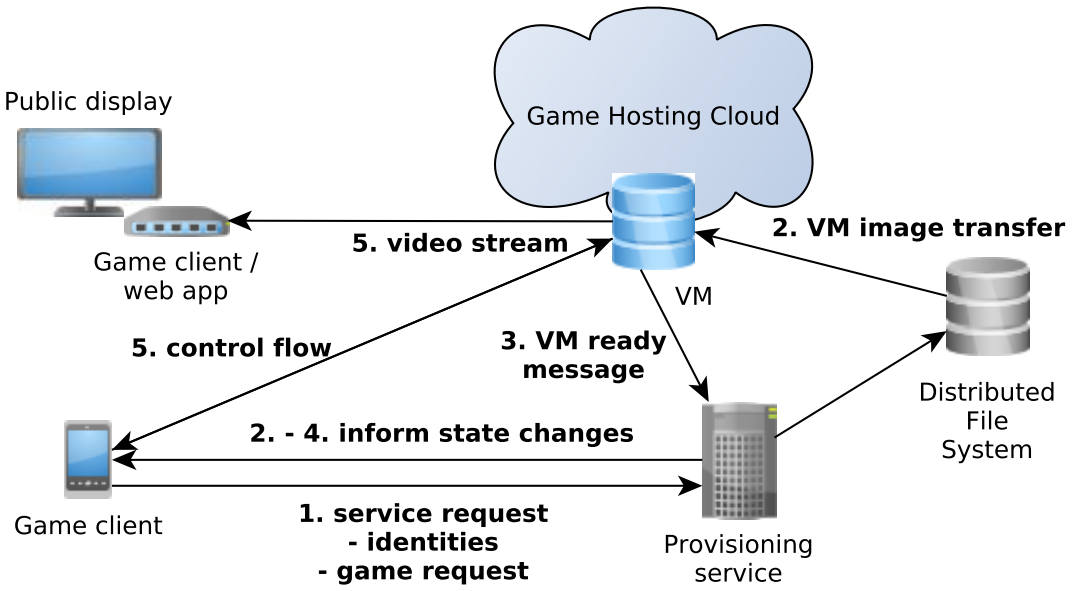

Figure 10: Workflow of initializing a game.

to the selected host. Meanwhile, the Resource Provisioning Server will inform the gaming client that the game is being initialized and the procedure may take a while. Alternatively the Resource Provisioning Server could initialize the game server on the furthest available server that can still meet the criteria thus reserving resources for other users. The optimization of the provisioning service is left for future work.

3) When the images have arrived on the selected host, a launch VM will be instantiated. The VM has the gaming server and its relevant software running on it. When the VM is launched, the state information will be forwarded to the gaming client through the Resource Provisioning Server.

4) The game client will initialize a connection with the launched cloud gaming server. The configuration of the gaming server, including video resolution, frame rate, and the control over video and control flows, may be adapted to the computing and networking environment.

5) When the player starts a game session, the control flow is sent to the gaming server and the video stream is delivered back to the gaming client. If a user wants to utilize a public display, he/she can for example enter a specific URL to the public display, which opens a web application containing a unique QR code. The mobile device can read the QR code through the gaming client. The gaming client then forwards the information about the public display to the Resource Provisioning Server, which sends the cloud server information to the web application open in the public display. The web app will then connect to the cloud gaming server and the user can start playing using the public display for video and the mobile device only for controls.

When the user stops playing the game and plans to move away, he can simply exit the gaming client on the mobile device, which will inform the Resource Provisioning Server to shut down the VM and free up the resources. If the user would like to pause the game and continue it somewhere else, the snapshots of the VM can be stored and resumed later. The snapshots have to be stored in the distributed file system as the cloudlets themselves only have a soft state.

4.6 Mobility management and security

The suggested design of the distributed mobile cloud gaming system could have some issues related to mobility management. For instance the mobile user might switch from a cellular network to a Wi-Fi connection during a gaming session, which would result in loss of connection if mobility aspects are not considered in the system. This switch could be handled in multiple ways. The simplest solution would be just to program the cloud gaming software to reconnect to the server when it recognizes that the connection has been lost. Another possible solution is to use protocols for mobility management such as the Host Identity Protocol (HIP). On top of providing mobility solutions it could also have other beneficial features.

HIP replaces the use of IP addresses as identifiers with special Host Identifiers (HI). By decoupling the internetworking layer from the transport layer it eases mobility management since the HI doesn't change even if the underlying network connections change. HIP has a built-in system to notify the other party when its

IP changes. As the applications use the Host Identifiers for connections, they don't need to know about network changes in the protocol layers below.

HIP can be configured to use IPsec for protecting the traffic between two hosts. This way the traffic between the cloud server and the user is encrypted and cannot be eavesdropped. The prototype built in the next chapter supports the HIP architecture although further evaluation is left for future work.

5 Cloudlet Remote Gaming Platform

This section outlines the design of the Cloudlet Remote Gaming Platform and the specifics of the built prototype. The changes made to the remote gaming software are further discussed and an HTML5 alternative that could be used for example in the public displays is also presented. Finally an additional module called the Virtual Controller Module is specified which will help in adding gamepad support both for the native application and the HTML5 alternative.

5.1 Prototype design

The Cloudlet Remote Gaming Platform is designed to be deployed anywhere between the distant cloud server and the user. Cloudlets can by definition be anything from a large data center to a single computer and so can the designed platform. The only limitation is that the implementation must have access to GPUs to be able to serve clients for gaming. The objective is that a cloudlet can serve any number of concurrent users limited only by the available computational resources.

A single VM of a cloudlet should also be able to serve any number of players limited only by the design of the particular game. The focus is on using mobile devices for the clients although the system supports PCs as well. The built prototype will be designed to be deployed at a private cloud in the user's home or other location with a Wi-Fi network. Nevertheless the prototype can be also used to evaluate other locations for the cloud servers and be accessed from anywhere from the Internet.

The core functionality is that users can instantiate a custom remote gaming enabled virtual machine using his/her mobile device. The mobile device discovers the cloudlet, either by itself or with the help of the resource provisioning server explained before, and instructs it to create a custom virtual machine. When the remote VM is up and running it starts to stream the game to the device and the user's device sends control information back to the VM. The software should also support the use of public displays and external controllers.

The built prototype must be able to launch virtual machines with GPU support. It must also be able to advertise its presence to mobile clients. The platform should support the VM synthesis explained before. The launched VMs should have the necessary remote gaming software running to which the mobile device can connect after the cloudlet informs the resource provisioning server which in turn informs the mobile device that the VM is ready. This enables the user to begin interactions with the running VM by connecting a remote gaming client to a server running on the VM. For controls the remote gaming software should support the touch screen of the mobile device or an external controller for example a gamepad. The main components of the platform prototype are presented in Figure 11. When the user is done playing the game, he or she exits the application on the mobile device, which should trigger the shutting down of the VM leading to the resources being freed for other users.

The platform prototype should be built from open-source components so it can be further developed and tested in the future. For proof-of-concept a small prototype

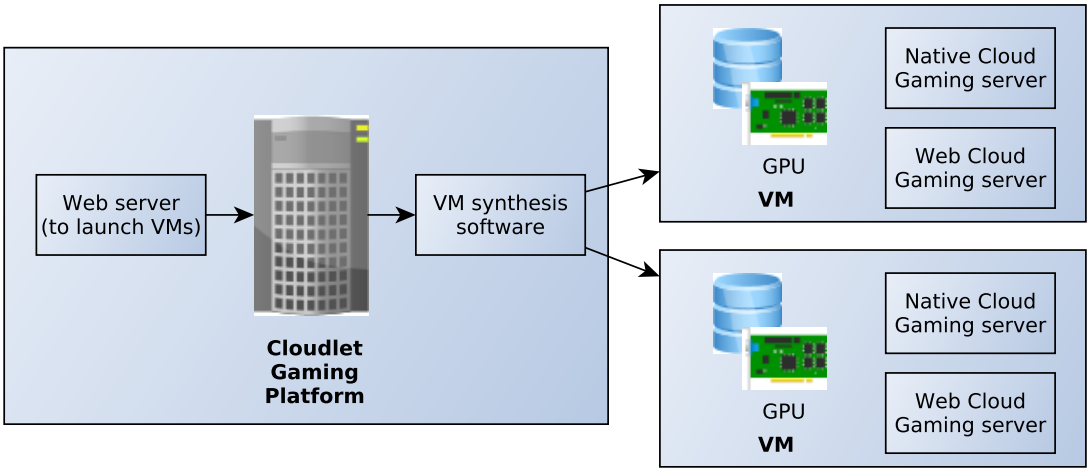

Figure 11: The main components of the Cloudlet Remote Gaming Platform prototype.

.

cloudlet capable of launching three VMs with GPU support at the same time will be constructed and evaluated. Each VM will support at least 4 players to play the same local multiplayer game remotely. The rest of the chapter focuses on the individual components that are needed to build the cloudlet prototype.

5.2 Hardware & Operating systems

In order to evaluate the feasibility of using cloudlets for cloud gaming, a Cloudlet Remote Gaming Platform prototype is built allowing users to play the latest and most GPU intensive desktop computer games remotely on almost any mobile device or PC. The games run on virtual machines (VM) inside the host operating system. Video and audio are streamed to the remote device from the cloudlet and game control commands are sent from the mobile device to the VMs.

The prototype platform consists of a single PC with Internet connectivity and multiple GPUs. The PC in the prototype implementation is a 2009 Apple Mac Pro with Intel Xeon CPU E5520, 19 GB RAM and three graphics adapters. For the host the machine has a GeForce GT120 and for the guest machines (VMs) an ATI Radeon HD 5500 and an ATI Radeon HD 7750. For the particular virtualization methods used in the prototype, any Intel VT-d, or AMD-Vi supported platform could be used. The platform is also able to utilize any number of graphics cards that can be fitted into the particular PC. Emulated graphics cards normally found on VMs do not perform well enough to run complex 3D games and therefore additional requirements must be set for the Cloudlet Remote Gaming Platform.

The host operating system (OS) is Arch Linux although any Linux distribution capable of utilizing the latest Linux kernel (v. 3.9 and upwards) virtualization features could be supported. The Elijah cloudlet software presented earlier is used for handling the cloudlet synthesis process. The original Elijah cloudlet software

uses a modified version of the QEMU machine emulator (v.1.0) to launch the guest VMs. This version does not support VGA passthrough so the Elijah modifications needed to be applied for a more recent version of QEMU (v.1.5). Unfortunately because of time limitations some of the special functionalities of the Elijah QEMU version were lost in the process. The original Elijah modified version can do the synthesis process for both the disk and the memory state of the VM. The modified version used in this thesis is currently unable to handle the memory synthesis. This can be bypassed using the operating system's own hibernation method to save the memory state into the disk. After this only the disk is used to perform the synthesis. This adds some overhead to the process and is something that should be fixed in future work. This fix is enough for now since the purpose of this thesis is not to speed up the synthesis process.

5.3 Monitoring and logging the delay

The open-source cloud gaming platform GamingAnywhere presented earlier is used for both the server and the client. The GamingAnywhere server and client are currently available for Linux, OS X and Windows and the client is also available for the mobile Android operating system. To find solutions for two of the issues presented in the problem statement of this thesis some modifications had to be made for both the GamingAnywhere (GA) client and the server. This is possible thanks to the openness and modular design of the GA software.

The assessment of the total response delay (RD) perceived by the player requires a way to measure the delay added by both the server and the client. The developers of the GA software have made some comparisons [21] of the performance of the GA compared to other cloud gaming platforms. These measurements have proven the GA to be extremely competitive against the available proprietary solutions. Unfortunately the GA software does not currently offer built-in ways to evaluate its performance. Luckily the GA is designed to be modular so injecting the needed time logging into the software can be done.

The processing delay (PD) which consists of memory copy, format conversion, video encoding and packetization needs to be measured on the server side. On the client-side one needs to measure the playout delay (OD) consisting of frame buffering, video decoding and screen rendering. The logging method is the same for all delay components. The function or functions responsible for the particular part of the process are wrapped with time logs to the beginning of the task and immediately after the result. The time difference is calculated next and the output logged. This is repeated for all frames processed giving the possibility to calculate the averages over a certain time period. The following code example explains the logging method.

```
timeval start, end, diff;
// Log the starting time of the process
::gettimeofday(&start, NULL);
...
// The process under measurement
...
// Log the ending time of the process
::gettimeofday(&end, NULL);
// Calculate the difference
timersub(&end, &start, &vdiff);
// Log the result converted to milliseconds
log("Delay: %lu ms\n", diff.tv_usec / 1000);
```

The logical modules of the GamingAnywhere system and the logged functions are listed in Tables 2 and 3. These logs are used in measurements in Chapter 6.

Table 2: Logged delay components of the processing delay (PD)

Delay component	Module	Logged part of code
Memory copy	vsource-desktop	Frame capturing while loop in vsource_threadproc(void *arg)
Format conversion	filter-rgb2yuv	Frame conversion while loop in filter_RGB2YUV_threadproc(void *arg)
Video encoding	encoder-video	Video encoding part of the while loop in vencoder_threadproc(void * arg)
Packetization	encoder-video	Packetization part of the while loop in vencoder_threadproc(void * arg)

Table 3: Logged delay components of the playout delay (OD)

Delay component	Module	Logged part of code
Frame buffering	native client	Packetization part of the function afterGettingFrame(...)
Video decoding	Android Java client	Video decoding buffer in decodeVideo(...)
Screen rendering	Android Java client	Video rendering part in videoRendererThreadProc()

5.4 Gamepad support

Problem 4 in the problem statement presented the limited controller options when playing PC games on mobile devices. PC games are generally played with either a gamepad or the keyboard and mouse combination. GamingAnywhere currently offers only virtual mouse and keyboard button emulation through the touch screen of the mobile device. The playability of games on the mobile device could benefit from the possibility of using an external gamepad for controls. For this reason gamepad support was developed for the GA Android client. In the implementation one can choose between three modes: virtual controller, relative mouse emulation and absolute mouse emulation.

The virtual controller mode uses a new server module explained in Chapter 5.6.1 in more detail. In this mode the host operating system creates a virtual gamepad device on the system after which the commands are directed from the gamepad connected to the client to the virtual device through the new server module. This way the gamepad can be used in all games that have built-in gamepad support. The two other modes are for games that don't have native gamepad support.

Both the relative and absolute mouse emulation modes emulate the movement of a connected mouse and the key presses of a keyboard. In the implementation the left analog stick of the controller emulates the arrow keys often assigned to movement in various games. The right analog stick emulates mouse movements and is often assigned to looking around or moving a cursor in games. The two different emulation modes are needed because games handle mouse movements in different ways. Some games display a cursor on the screen and the cursor stays in the place it was left when the mouse movement stops. This style is emulated by the absolute mouse emulation mode. In some games the mouse movements turn the character over and the often hidden cursor centers again when the movement stops. The following code explains the implementation of the two modes using the existing sendMouseMotion function of the GA client software.

```
1  // Asynchronous function called every 5ms
2  protected Void doInBackground(Void... arg0) {
3     // Relative mouse emulation mode - move mouse to the direction of ↩
          the axis at a certain speed
4     if (!absoluteMode && (Math.abs(joyZ) > 0 || Math.abs(joyRZ) > 0)) ↩
          {
5        moveMouse(speed*joyZ, speed*joyRZ);
6        sendMouseMotion(mouseX, mouseY, speed*joyZ, speed*joyRZ, 0, ↩
             true);
7        lastX = lastX+speed*joyZ;
8        lastY = lastY+speed*joyRZ;
9        return null;
10    }
11    // Absolute mouse emulation mode - cursor position is the same as ↩
          the axis position
12    else if (absoluteMode) {
13       mouseX=getViewWidth()/2 + joyZ*getViewWidth();
14       mouseY=getViewHeight()/2 + joyRZ*getViewHeight();
```

```
15          if ((mouseX != 0) || (mouseY != 0))
16              sendMouseMotion(mouseX, mouseY, mouseX-lastX, mouseY-lastY, ↩
                    0, true);
17          lastX = mouseX;
18          lastY = mouseY;
19          return null;
20      }
21  }
```

5.5 Control and video channel separation

Problem 3 regarding playing PC games on mobile devices listed in the problem statement of this thesis is the small screen size which could limit longer gaming sessions or worsen the gaming experience. To overcome this issue one could switch to use an external display when possible and only use the mobile device for controls. The use of an external display might also save power on the mobile device. To make this possible, the control and display logic must be separated in the Android GA client. The GA client already uses separate threads and socket connections for the control and video signals. So the only changes needed to be made were to the logic of the client so it won't start the video receiving thread if the mobile device is only used for controls. This changes the interaction between the server and the client depicted earlier in Figure 6. Now a separate client connected to the public display or smart TV receives the video stream and the mobile device is only used for transmitting the input commands. The new interaction is presented in Figure 12.

The client of the mobile device can at any time start or stop receiving the video stream. This enables the user to switch between displaying the game image on the screen or on an external display when available. In addition to improving the game experience, the saved energy on the mobile device could enable one to play the game longer without the battery running out. The power measurement results are presented and analyzed in Chapter 6.

It might not be always possible to install a native gaming application on public display platforms or smart TVs. For this reason a HTML5 alternative capable of running in a browser needs to be developed.

5.6 HTML5 Alternative

The native GA client and server presented earlier in this thesis can be utilized in public display systems and smart TVs if the gaming client can be pre-installed or installed on the fly before initializing the game. This may not always be feasible since the public display system might be closed and controlled by a separate instance.

The interface of public displays is usually implemented through a browser. Therefore to be able to utilize clouds for remote gaming in public displays one can leverage the browser to run the client application. For this scenario an HTML5 based server and client were developed as an alternative for the native software. The browser-based client could also help in overcoming the problem of different platforms in

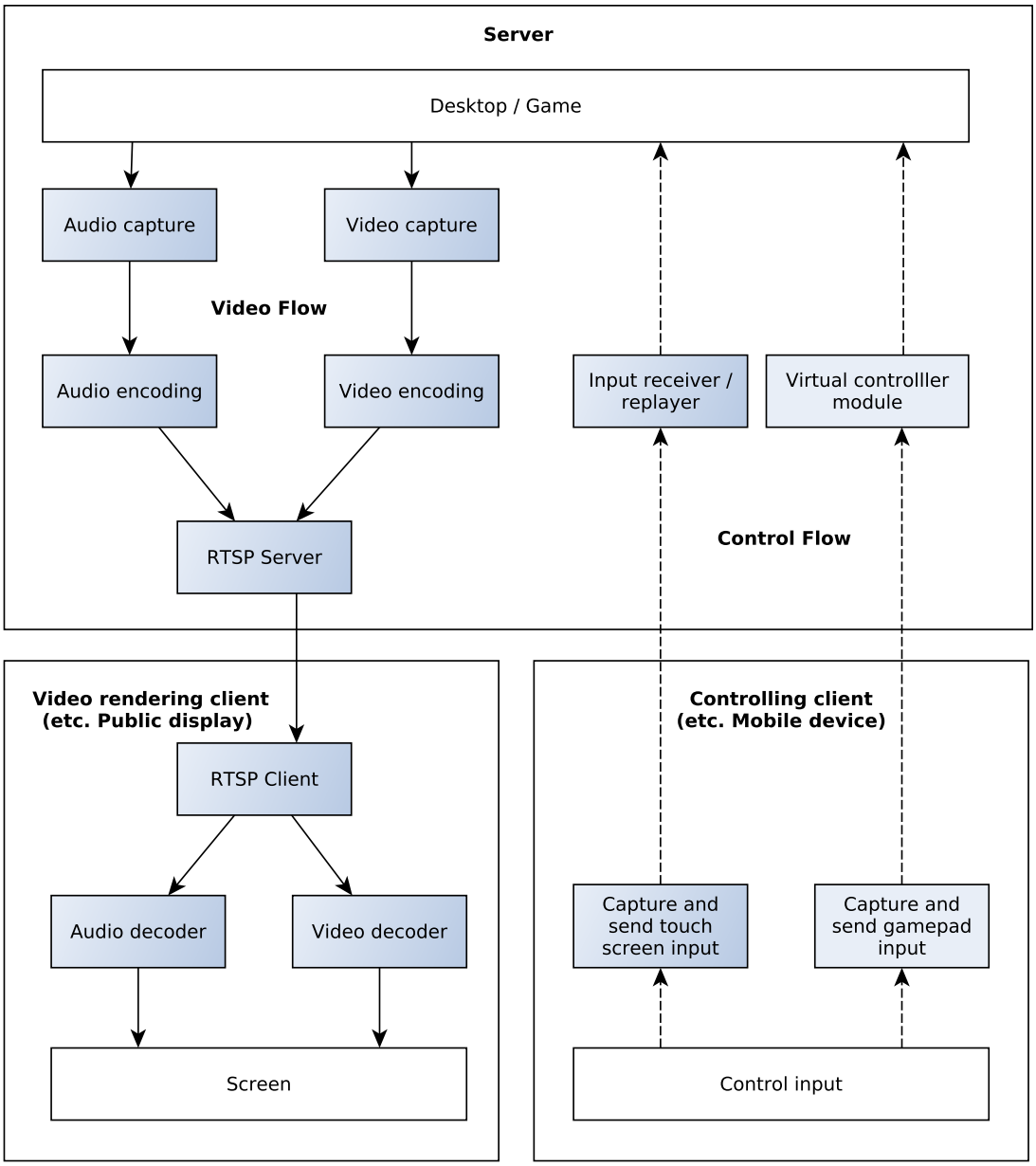

Figure 12: The main components of the Cloudlet Remote Gaming Platform.

.

mobile devices. All mobile platforms have a native browser implementation and at least some level of HTML5 support. This would also help on building a client application for smart TVs since their platforms also vary a lot. A simple web browser could be enough to stream the game screen to a smart TV.

The HTML5 alternative built is a web application constructed with Node.js[10] and FFmpeg[11]. Node.js is a platform built on Chrome's Javascript runtime. It is event-

[10]Node.js website: http://www.nodejs.org
[11]FFmpeg website: http://www.ffmpeg.org

driven and non-blocking, which makes it suitable for real-time applications. FFmpeg is an application for converting and streaming video. It includes the libavcodec audio/video library that is also used in the native GA software used in the other scenarios of the Cloudlet Remote Gaming Platform. Thus the performance of the server side logic of the HTML5 Alternative should be similar to the native one. The different components of the HTML5 Alternative are presented in Figure 13.

The logic of the web-based alternative is as follows. When a new client connects and requests a specific URL, the server establishes a web socket connection to the client browser and assigns a unique ID for the client. The client in turn asks for a video file from the server using the unique id. The video file on the client side is actually a live video stream from the virtual machine and is implemented with the standard video tags of HTML5 using the MP4 container and x264 video encoding.

When the video request comes in, the web server launches an FFmpeg process that captures the desktop and encodes the video into a fragmented x264-encoded MP4 file. The output of the process is directly piped into the HTTP response in real time. The client's browser starts receiving the file (video stream) and playing the stream immediately.

The mobile device and the public display or smart TV also need a way to identify themselves so that the same game streaming video can be displayed either on the external display or on the display of the mobile device. For this purpose a special central web service can be set up on a predefined URL. The URL can be entered to the browser of the public display or smart TV. This establishes a web socket connection between the browser and the server and also displays a QR code on the browser that can be read from the mobile remote gaming application. The QR code contains an identifier of the particular web socket used between the central web server and the public display. After reading the code, the mobile device can send the web socket ID and the game streaming URL to the centralized server. The server then contacts the browser of the external display with the correct game streaming URL after which the browser can start streaming the game video.

The HTML5 alternative also supports gamepad controls. Player's commands are captured with the browser using the GamePad API [9] developed by the W3C. The captured commands are formatted to JSON and sent to the web server through the web socket. On the server side the Node.js web server forwards the commands to the new Virtual Controller Module. The module receives the input commands, creates new virtual controllers when necessary and forwards the commands to the virtual devices. The virtual controller module is explained in Chapter 5.6.1.

It is often necessary in PC games to be able to move the mouse and access at least the most common keyboard buttons such as the cursor keys, enter and escape. For this purpose the client can activate keyboard and mouse emulation mode by pressing a certain button on the gamepad. When in mouse and keyboard emulation mode, the server re-uses the GamingAnywhere server by converting the control messages into GA's control message format and passing the messages for it.

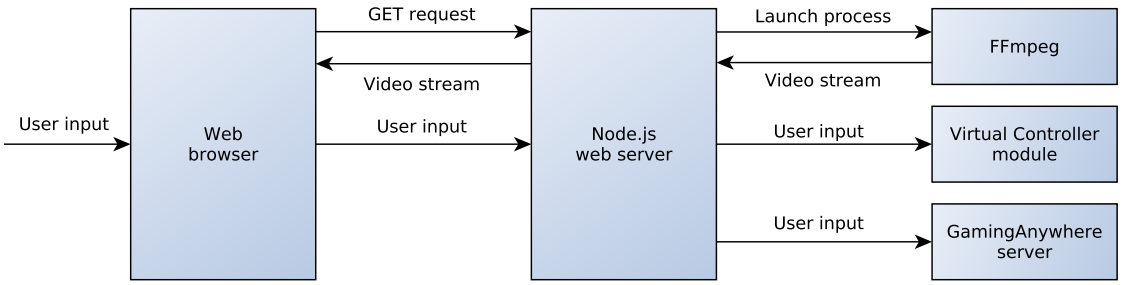

Figure 13: The components of the HTML5 Alternative for the Cloudlet Remote Gaming Platform.

.

5.6.1 Virtual Controller Module

PC games are usually played with either gamepads or the keyboard and mouse combination. The current version of GamingAnywhere (0.7.4) does not include gamepad support and only includes controller support for mobile devices by emulating mouse movements and keyboard presses with the device's touch screen. Furthermore the system currently only supports one controlling player although other players could watch the game being played. Gamepad support was added for the Android client earlier in Chapter 5. By reusing the GA control logic one can emulate mouse movements and keyboard presses using an external controller. Multiplayer support and virtual gamepad emulation requires an additional module to be built.

The modular design of GA makes it possible to add components to the system written in virtually any programming language as long as the modules can communicate with each other. The built virtual controller module is a Python-based server module. It receives control information from the web server in the HTML5 alternative case or directly from the Android client in the native client case. The client modules and the Virtual controller module format their messages using JSON. The messages are sent through TCP sockets. The server waits for new socket connections in a predefined port. The logical flow of the designed module is presented in Figure 14.

In the native client the gamepad movements and key presses are captured using Android's built-in methods to capture motion events. In the HTML5 alternative the web server forwards the control inputs captured by the client browser to the Virtual controller module. The Android client or the web server formats the control events into JSON objects and sends them to the module. For example

```
{
  "client": 1,
  "gamepadId": 1,
  "msgtype": "button",
  "button":"button-2",
  "pressed": 1
}
```

is a control message from a client with an ID value of 1. It tells that the button 2 of the client's first gamepad was pressed. When the virtual controller module receives an input message it first check if the given virtual controller exists in the OS and creates it when necessary. The amount of buttons and axes in the gamepad are predefined in a configuration file. By default the module mimics an Xbox 360 controller. The input command is then directed to the created virtual controller after which the operating system is responsible of handling the input. Games see the devices as real controllers.

The current implementation of the module works only on Linux based operating systems. The created module uses a Python module called evdev [7] that provides bindings to the generic input event interface in Linux. Evdev is able to create and handle input devices that can inject events directly into the input subsystem [7]. The designed module can support any number of controllers needed by the game. This adds remote multiplayer support for the Cloudlet Remote Gaming System in games that have local multiplayer support. This enables new multiplayer scenarios for the system. One could for example invite other players to join a game already running on a cloudlet initialized by the user. The other players could use their own mobile devices or any other device capable of running the remote gaming software or the HTML5 alternative. In addition multiple game controllers could be connected to the same mobile device enabling multiplayer support also through one mobile device.

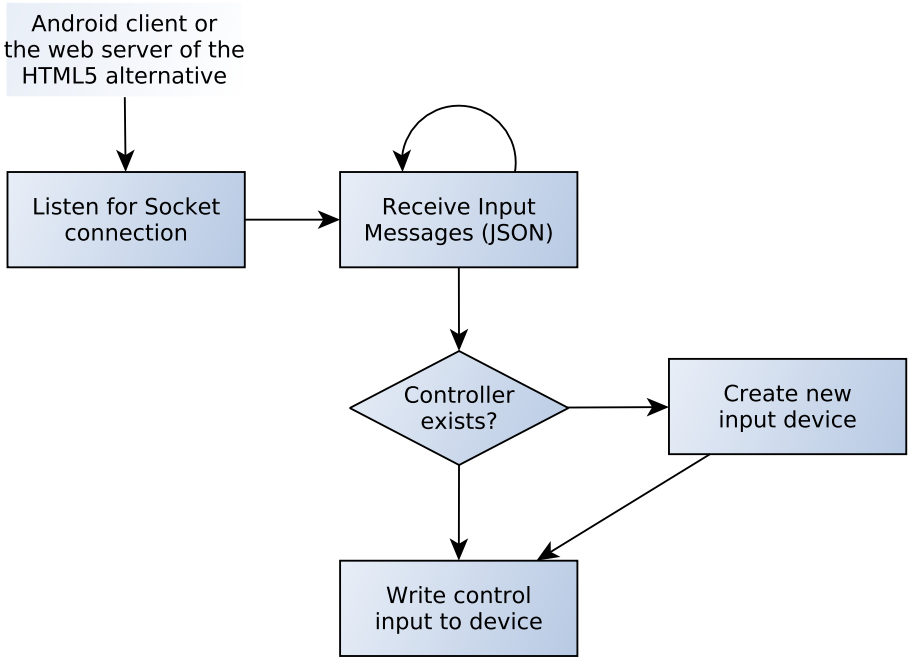

Figure 14: Flow chart of the Virtual controller module

6 Evaluation

This chapter analyzes if running remote games in cloudlets instead of a distant cloud could benefit the most latency-strict games by lowering the network delay part of the overall response delay. The benefits in power consumption when using an external display such as a public display or smart TV in the distributed cloud gaming system are also measured. In addition the performance of the HTML5 alternative compared to the native GA software is also tested.

6.1 Response delay measurement setup

The prototype implementation is evaluated in different scenarios in order to draw conclusions with respect to resulting quality of experience for different cloud gaming server locations. The focus is first on the response delay, which is the total delay between the user giving an input and observing the outcome of that input on the screen. It is arguably the most important performance metric as it directly reflects the quality of the gaming experience [24].

The varied parameters for all of the test cases are listed in Table 4. For the delay measurements the network type and server location were changed. The client was connected to the Internet using either a Wi-Fi or a dedicated campus LTE network that was very lightly loaded. The gaming platform was deployed either locally or in a remote cloud. Local deployment included two cases: When using Wi-Fi access, it was deployed in the same local network as the mobile device. When using the LTE access, the gaming platform was deployed behind a fiber connection from the Internet Service Provider (ISP), which in practice provides similar latency as if it was deployed within the ISP's network. The remote deployment was at the Amazon EC2 cloud service. The closest available location available was chosen, which is in Ireland. In the Amazon EC2 deployment case, a GPU instance with one Nvidia Grid GPU (Kepler GK104) was used, whereas the local deployment consists of the Cloudlet Remote Gaming Platform described earlier running in a single PC with multiple GPUs.

For the different deployment scenarios and networks the overall response delay was measured and for the control options the power consumption of the mobile device was also measured. In the delay measurements a Samsung Galaxy Tab 3 (SM-T315) tablet was used. Each test case consisted of 10 minutes of repeatedly

Table 4: Parameters varied in the different test cases

Parameter	**Values**
Network type	Wi-Fi / LTE / 3G
Server location	Local (ISP) / Remote cloud (Amazon EC2)
Control	Touch screen / External gamepad
Display	Smartphone / External display

playing one level of a game called Trine 2, which ensured that the control input rate and the rendered graphics were similar between the different test cases and provided comparable measurement results. The screen resolution was set to 1280x720 with 60 frames per second in the tests. Furthermore the desktop capture mode of GamingAnywhere was used which is more compatible than the possibly more efficient game-hooking mode. The rest of the configurable parameters were set to the recommended values from the work of Huang et al. [21]. The GamingAnywhere options used are presented in Table 5. The periodic capture mode was used to achieve better compatibility with different games.

The video parameters used in the measurements are depicted in Table 6. The same settings were used both in the GamingAnywhere software and the HTML5 alternative when applicable. In the HTML5 alternative the VP8 codec was also used in browsers not supporting x264.

In order to measure the total response delay (RD) perceived by the player, the modified GamingAnywhere software injected with timestamps to different parts of the code was used. In this way, it is possible to log and analyze also the breakdown of the total RD into delay caused by the client and server-side processing in addition to the delay caused by the network. Specifically, the measured properties were the network delay, the server-side processing delay (PD) that consists of memory copy, format conversion, video encoding and packetization, and the client-side processing delay (OD) consisting of frame buffering, video decoding and screen rendering. Memory copy refers to capturing the raw image from the game or desktop, format conversion is the conversion of color-space, and frame buffering refers to the reception of all the necessary packets for one video frame.

The measurement is repeated for all frames processed giving the possibility to calculate the averages over a certain time period. The network delay was measured using the ping tool that measures the network round-trip-time (RTT) in specified time intervals.

6.2 Response delay measurements

The Cloudlet Remote Gaming Platform was deployed in three distinct locations. First it was deployed to a distant cloud server running on the Amazon EC2 cloud service. For the second experiment the cloudlet platform was moved to simulate the conditions of running the platform at operator premises. For this an LTE test

Table 5: GamingAnywhere options used in the measurements.

GamingAnywhere options	
enable audio	false
protocol	TCP
control protocol	UDP
capture method	ga-server-periodic

Table 6: Video parameters used in the measurements.

General settings	
video encoder	libx264
FPS	60
bitrate	4500000
frame references	1
motion estimation method	dia
motion estimation range	16
key frame interval	48
slices	4
threads	4
x264 specific settings	
profile	main
preset	faster
tune	zerolatency
intra-refresh	1

network setup called Netleap in Otaniemi, Finland was used. The cloudlet had a fiber connection to the operator premises of the test network so the server is practically at the operator premises. A difference of 1-2 milliseconds was observed when measuring the latency to the cloudlet server compared to the first pingable instance behind the packet core of the LTE network from the mobile device. Finally the cloudlet was moved to be connected to the same local network as the mobile device. This scenario used a Wi-Fi connection between the mobile device and the cloudlet.

The results of the latency measurements are depicted in Figure 15. The playout delay was fairly consistent with values between 10 and 15 milliseconds. The mobile device handled the video processing well thanks to its built-in hardware video decoder. As expected, the network delay dominates the overall delay in the case of non-local deployment. Using a 3G connection the network delay averages just below 100 ms, while switching to an LTE network reduces the network delay by one third resulting to 63 milliseconds. The processing delay was measured to be roughly 20 ms for both the local deployment and the remote Amazon EC2 deployment. This shows that the prototype's virtual machine is fairly similar in processing power compared to the remote cloud's instance.

Comparing the overall response delay in the local vs. remote cloud deployment, the difference is large, from 50 ms to 100 ms when using LTE, because of the relatively large differences in network delay. These results highlight the need for distributed cloud deployment for certain types of games. As discussed earlier, the games with the strictest requirements may demand the delay to be no higher than 60 ms before the user experience begins to be affected. Based on measurements it can be seen that only the local Wi-Fi and the local LTE deployment in operator

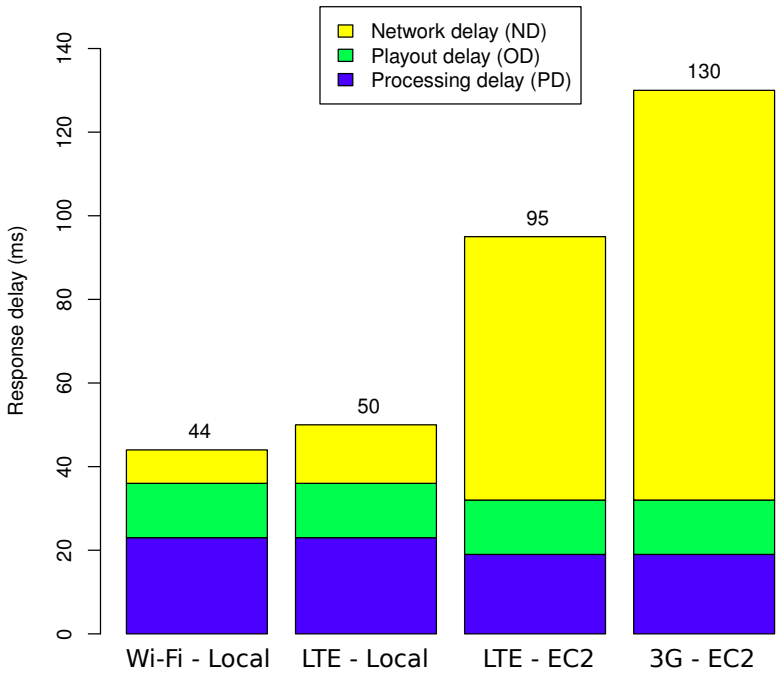

Figure 15: Response delay in different scenarios.

premises could be able to fulfill the latency requirements of the most demanding fast-paced games.

For medium-paced games the threshold has been previously defined to be around 130 ms. The LTE scenario to the distant Amazon EC2 cloudlet fulfills this requirement with a delay of 103 ms. The 3G measurements go slightly over the threshold for medium-paced games and should only be used if no alternatives are available. All deployment scenarios are able to fulfill the 190 ms requirement of slow-paced games.

The results show that there's a need for the proposed distributed cloud gaming system. The most demanding games require the cloud server to be deployed closer to the user. On the other hand, many types of games do not require such a low latency because of which the dynamic provisioning in the proposed distributed cloud gaming system becomes very useful. Games with low latency requirements will be deployed closer to the user, while games with less stringent needs can be deployed in a more centralized manner on more distant cloud.

The measurement results so far however only take into account the delay when playing a single-player game or a local multiplayer game on the remote cloud server. Online multiplayer games are massively popular and connecting to such games adds an additional delay, which should be taken into account. The typical latency across a continent is around 50 ms. This yields to an average of 25 ms if the multiplayer

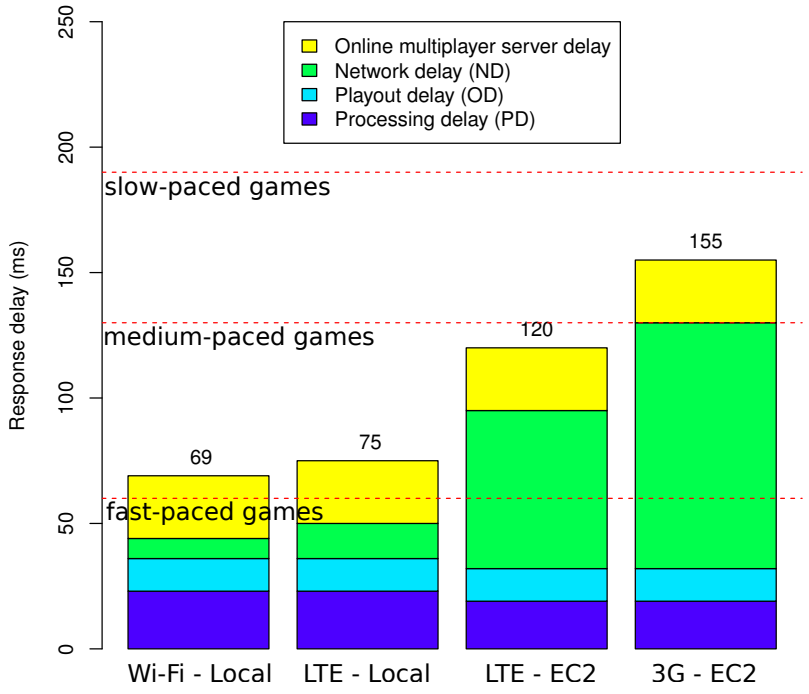

Figure 16: Response delay in different scenarios with added online multiplayer server delay.

server resides on the same continent as the cloud gaming server. Adding this to the overall response delay puts even the local deployment scenarios in trouble regarding the most demanding fast-paced games. This is not so clear though since the existing delay compensation mechanisms can be applied to mitigate the delay between the rendering cloud server and the multiplayer server. However extra delay caused by multiplayer online games should be taken into account when deciding the location of the cloud gaming server. Online multiplayer games should be prioritized to be run on closer servers than single-player or local multiplayer games. In some games with known centralized online gaming servers it might be even justifiable to launch the cloud gaming server on a location close to the game's own online servers. Figure 16 shows the final results with delay of the online multiplayer added to the response delay caused by the cloud gaming server. The thresholds for fast-, medium-, and slow-paced games are also drawn to the figure as dashed horizontal lines.

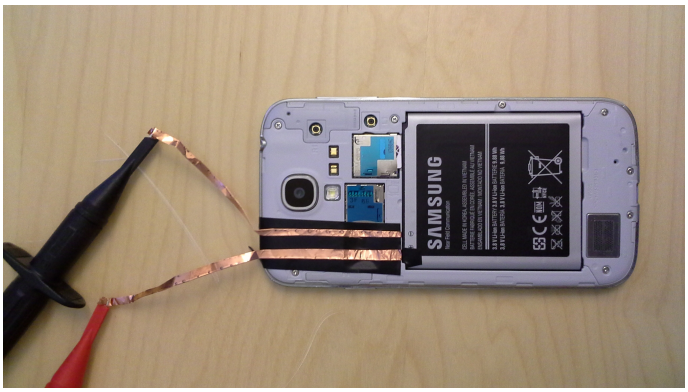

Figure 17: Bypassing the battery of the mobile device.

6.3 Power measurement setup

For the power measurements the mobile device was connected to a Monsoon Power Monitor[12] that powers up the device and measures its power consumption at the same time. The power monitor logs the power consumption of the mobile phone several times a second giving both the opportunity to visualize the power consumption over a time period and also calculate the average power consumption during a game session.

The mobile phone (Samsung Galaxy S4) was connected to the Monsoon Power Monitor by bypassing the battery of the device. The voltage terminal of the battery was covered with insulating tape and copper foil tape was used to allow connecting the Monsoon Power Monitor device to the phone. The setup shown in Figure 17 allows the device to continue communicating with the battery as only the voltage and ground terminals are bypassed. Software called PowerTool was used to record the data measured by the power monitor and to export the data for statistical analysis. An overview of the measurement setup is shown in Figure 18.

The control and display parameters of Table 4 were changed between the test cases. The game was controlled either by using the touch screen or by connecting an external gamepad to the mobile device. The game was visualized to the player on the mobile device screen or on a separate display.

6.4 Power measurements

Battery life is a crucial factor in today's mobile devices. Video streaming and the continuous network connection in cloud gaming could drain the battery quickly. Hence, a secondary evaluation target in this thesis is to quantify the power consumption of the mobile client when using the cloud gaming platform. The added power consumption of cloud gaming vs. a native game running on the mobile device is first measured to see how cloud gaming affects the power usage of mobile devices. Next the evaluation continues by trying to reduce the power consumption of the cloud gaming session by using the new control and display options enabled by the

[12]Monsoon Power Monitor website: http://www.msoon.com

new system design. The use of a public display or a smart TV could possibly save a lot of energy since the mobile device no longer needs to stream the video and can therefore keep the screen dimmed.

Figure 19 shows the power consumption difference between a native game being run on the mobile device and the same game being streamed from a cloud server with the GA remote gaming software using an LTE network. The game OpenArena was chosen for this test since it is available both for Linux and the mobile Android platform. The cases of online and single-player gaming were separately measured to see the effect of the network usage on power consumption.

The results show that in single-player games cloud gaming doesn't necessarily save energy. Although the mobile device doesn't have to do heavy computational tasks in cloud gaming, it does need to constantly stream the game video from the network. The heavy effect of the radio transmitters on power consumption can be seen from the native online multiplayer gaming measurement results. Although the data amounts are quite small, the device still needs to keep it's radio transmitters on which proves to count for a large amount of the overall power consumption. The native online case consumes energy the most as it needs to run the game software as well as use the radio transmitters for communication with an online multiplayer server. The cloud gaming case consumes less energy than native online although more than the native offline case. It can be concluded that in online multiplayer games cloud gaming can save energy on mobile devices. In the case of single player games the benefits come from other advantages of cloud gaming. This includes running games on the mobile device that couldn't be normally run on the device because of platform or computational limitations.

The evaluation is continued by measuring the effect on power consumption when switching from an LTE network to a Wi-Fi connection. In both cases the power consumption is also measured with and without the use of an external display. This simulates the use of a public display in the system design. The average power measurement results are shown in Figure 20 and the continuous power consumption in Figures 21 and 22.

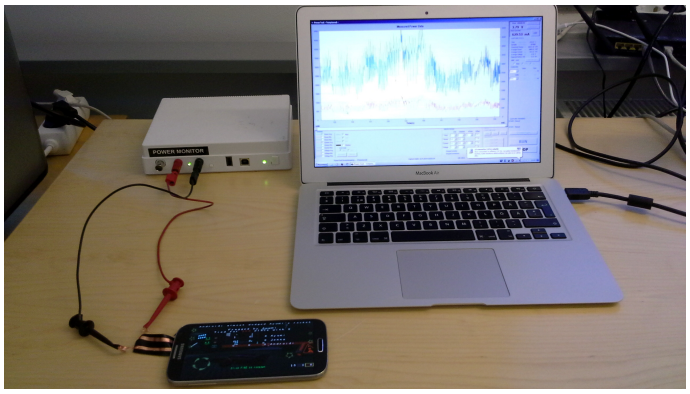

Figure 18: Power measurement setup.

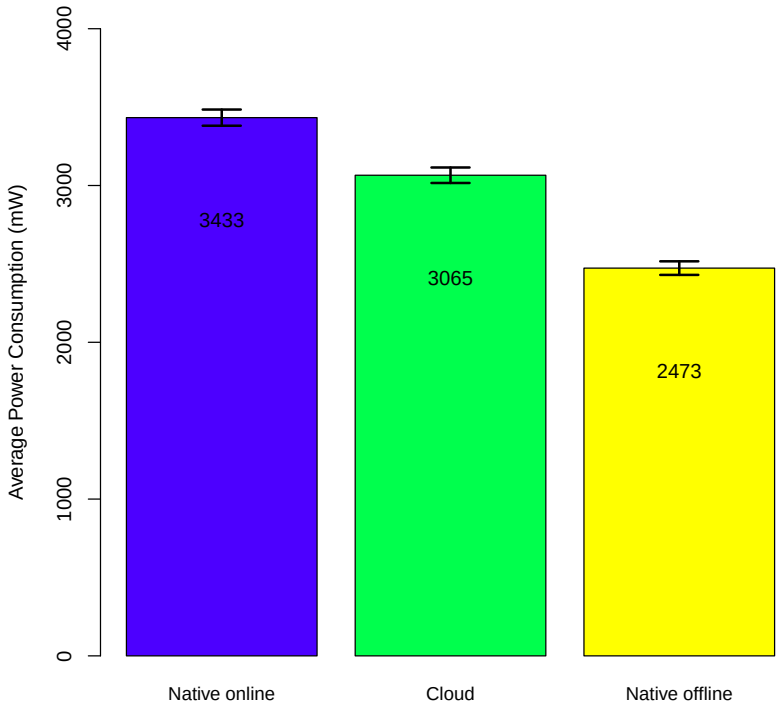

Figure 19: Average power consumption in online and offline native gameplay vs. cloud gaming.

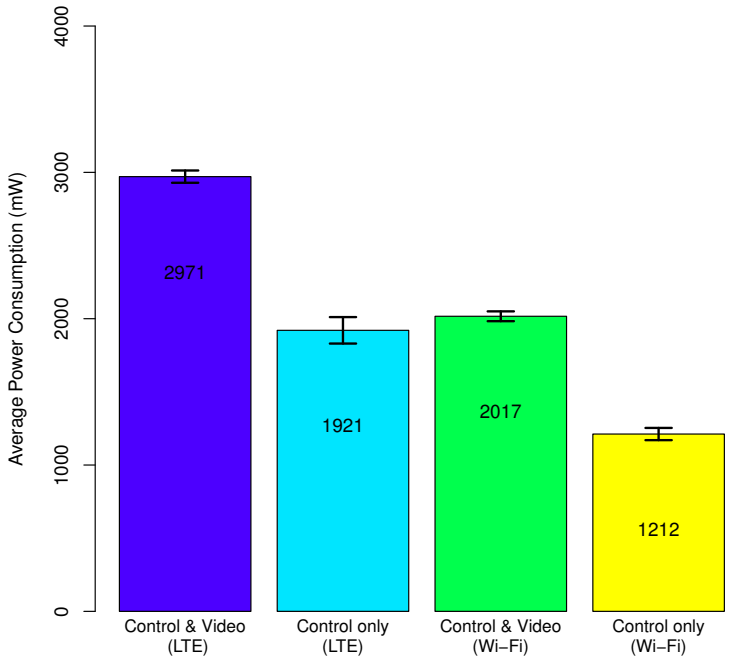

Figure 20: Average power consumption with and without video rendering on the mobile phone using LTE and Wi-Fi networks.

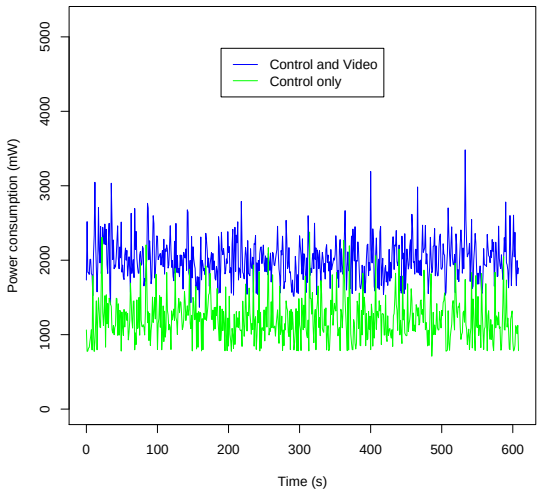

Figure 21: Power consumption, WiFi connection.

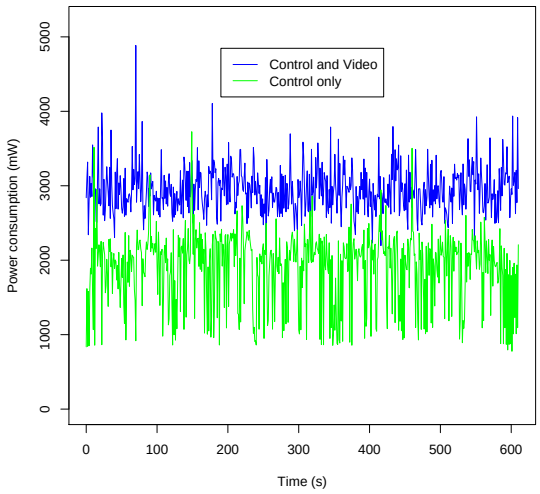

Figure 22: Power consumption, LTE connection.

Comparing first the impact of the access network technology, gaming using the LTE access and having both controls and video by the mobile device draws almost 3 W of power, which yields about 3h15min of battery life with a fully charged phone. Using Wi-Fi cuts down the power consumption by a third. This result is logical because it is well known that cellular network access exhibits a relatively large amount of so-called tail energy [10, 22]. DRX was enabled in the LTE network but it seems not to help that much because of the constant stream of incoming video data that prevents DRX from triggering. Wi-Fi exhibits a more linear scaling of the power draw as a function of the actual data rate [35].

When using the mobile phone only for control purposes with an external display visualizing the game yields very significant energy savings: 35% in the LTE case and 40% in the Wi-Fi case. This observation validates the presumption that the mobile gaming time can be significantly prolonged by using an external display for playing the games and using the mobile device only for forwarding control commands. The causes are two-folded: first, the screen of the mobile device can be dimmed in the modification of the gaming client when the mobile device is used only for controls, and, second, the device saves energy in wireless communication and computing by not having to receive and decode the incoming video stream in the control only mode. The test case using Wi-Fi access and external display delivers a battery life of almost 8 hours.

The results presented in Figure 20 were achieved using a gamepad as an external controller. For comparison the power draw when using the touch screen for controlling the game was also measured. Figure 23 shows that the average power consumption grows by approximately 500 mW when using the touch screen instead of the external gamepad. The variation in power consumption also grows when using the touch screen because the external gamepad is powered constantly (over

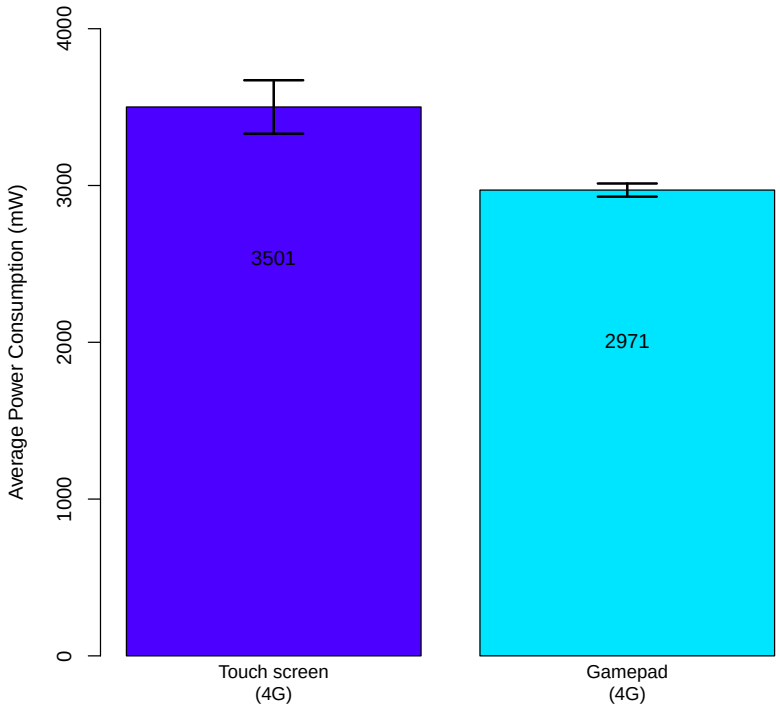

Figure 23: Average power consumption, gamepad vs. touch screen.

micro USB), whereas the touch screen generates power spikes when the screen is touched. Overall the power consumption in the most demanding scenario (LTE, control & video, touch screen) compared to the least demanding one (Wi-Fi, control only, gamepad) is almost three times high.

6.5 HTML5 alternative and Native client performance comparison

The performance of the built HTML5 alternative was highly dependent on the platform and browser used for testing. Measuring its performance was also trickier than for the native client as logging the frames drawn by the browser proved to be difficult. The capture times of the ffmpeg module could be logged in the same manner than in the GA case. However logging the drawn frames of browser seemed to hinder the overall performance of the browser. In the end the performance of the HTML5 alternative was measured manually by running the server and the client on same machine. A stopwatch application was launched on the host machine and the response delay was measured by taking screenshots of the monitor while the browser client was running beside the real-time stopwatch. The time differences of the stopwatch application were calculated next and the results were averaged.

Google's Chrome desktop browser performed the best. This is because it sup-

ports playing back x264-encoded videos in fragmented MP4 container with the so-called moov-atom in the beginning of the file. For example Firefox doesn't support x264-encoded videos. For this a less efficient VP8 encoder had to be used together with the OGG container. All mobile browsers tested also only supported the less efficient VP8-encoded video. Although some of the mobile browser's do support x264 videos, they don't support fragmented MP4 files which was a requirement for the system to stream the video without sending the whole capture video first. The moov-atom has information about the video, which is needed by the browser before it can start playing back the file. The desktop version of Chrome was the only browser supporting this feature.

The HTML5 performance results against the native GA client are shown in Figure 24. The desktop version of Chrome was the only browser with a usable response delay. The calculated 140 ms is quite high but shows the potential of using browsers also for cloud gaming. The delay difference to the native client is likely due to unavoidable buffering in the browser's programming. The native client draws a frame instantaneously after receiving all the packets belonging to a video frame. The browser however is optimized for non-interrupted video playback. All mobile browsers and the desktop Firefox browser suffered from the poor performance of the VP8 codec in real-time encoding.

The gamepad API used in the HTML5 alternative implementation had compatibility issues as well. Currently only the desktop versions of Firefox and Chrome support it. This is understandable since the API specification is still under development. This compatibility issue should be fixed in the future when the final version

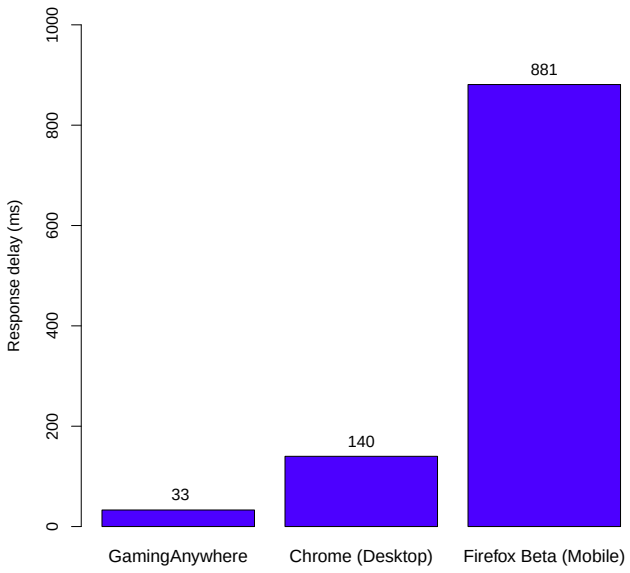

Figure 24: Response delay of the HTML5 alternative against the native GamingAnywhere software.

of the API is released.

6.5.1 Use of computational resources

The CPU & GPU load incurred is important for both the server and client side. The server side load states how many games and video streams can be played and streamed for clients. The remote gaming software should use as little computational resources as possible to leave most of the processing power for the games. The resources used by both the native and HTML5 server and client are compared next. For these tests the game Little Big Racing was played remotely on both a mobile device and a PC. The same level was repeated again for consistent results between test cases. The CPU load was logged with the Unix sar (System Activity Report) tool on PCs and with the Trepn Profiler tool on Android. GPU usage was logged on PCs with the nvidia-smi tool provided by the graphic card drivers. On the mobile device the Trepn Profiler tool logged the GPU usage as well.

Figure 25 shows the server side CPU usage both with the native GamingAnywhere software and the HTML5 alternative. As expected the resource usage is very similar since the game itself is the same and both approaches use the same ffmpeg libraries to encode the video stream. The GPU usage shows similar results in Figure 26. The more powerful GPU 1 (ATI Radeon HD 7750) handled the game and video encoding better in both test cases. The older GPU 2 (ATI Radeon HD 5500) was at its limits but was still able to handle the game and the encoding process.

It is notable that the CPU usage is fairly low even with two game instances and video encoding processes. This shows that the added GPUs help the virtual

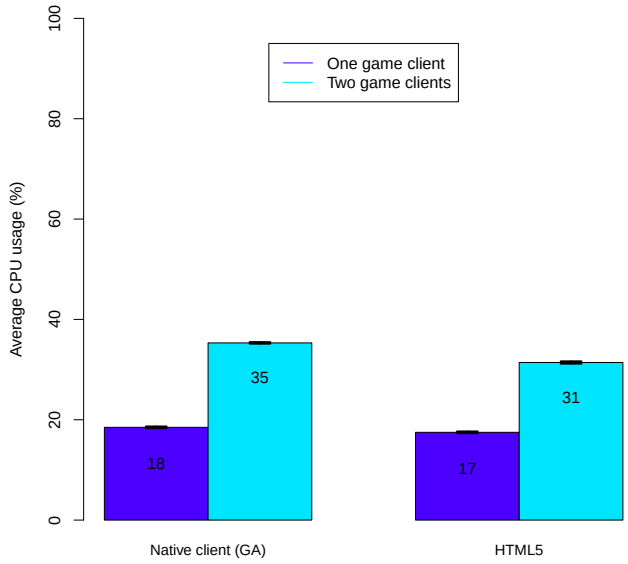

Figure 25: CPU usage comparison, Native client (GA) and HTML5 alternative.

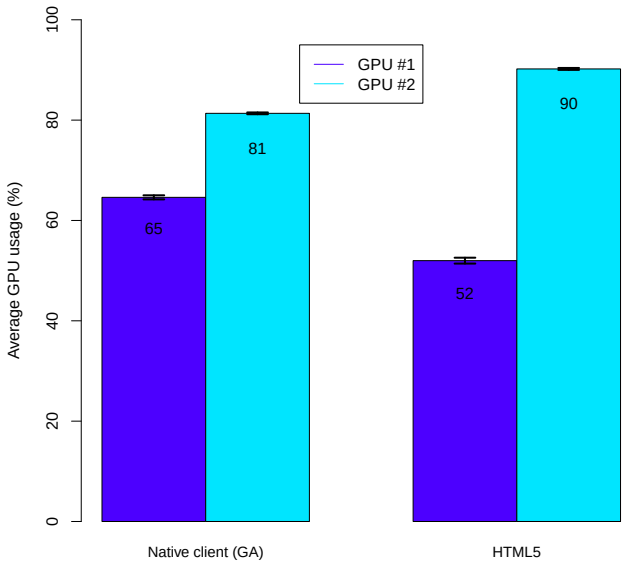

Figure 26: GPU usage comparison, Native client (GA) and HTML5 alternative.

machines to run the games even with one CPU. The system and the CPU could handle more clients with the aid of additional GPUs.

On the client side the CPU and GPU use were logged with both a PC and a mobile device. On the relatively powerful PC (Intel Core i5, nVidia GTX 680) the CPU and GPU use were quite low both on the Native client and the HTML5 alternative. The browser-based HTML5 alternative used slightly more resources with 18 and 13 percent average loads compared to GA's 8 and 8 percent average as shown in Figure 27.

The mobile device (Samsung Galaxy S4) recorded an average CPU use of 14 percent and 61 percent for the GPU using the native client. This shows that the mobile device utilizes well the GPU for the decoding of the video. The CPU and GPU use for the HTML5 use case were more even with an average of 33 percent for the CPU and 41 for the GPU. The difference can be explained by the fact that for the mobile device the HTML5 alternative has to use the VP8 codec that seems to incur more load on the CPU. The resource usage for the mobile device is presented in Figure 28.

Overall the resource usage of the native and the HTML5 client and server were fairly similar excluding the mobile device case where the different codec makes the comparison difficult. This shows that browser-based cloud gaming is a promising field if the compatibility issues regarding the video codecs and the game controllers are solved.

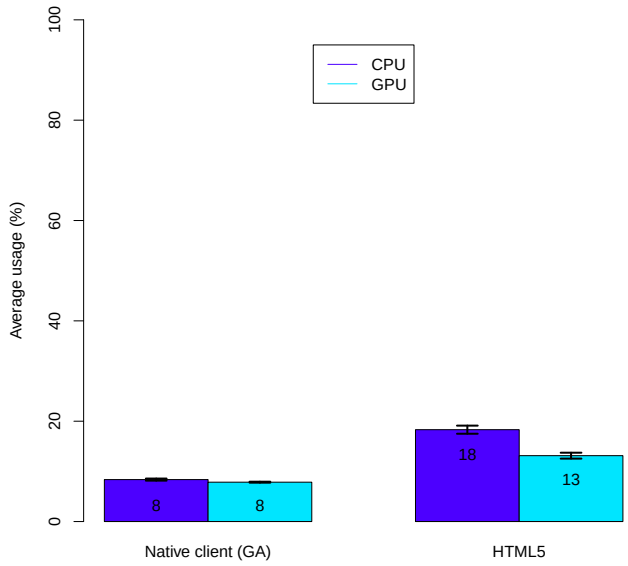

Figure 27: Client side PC CPU and GPU usage comparison, Native client (GA) and HTML5 alternative.

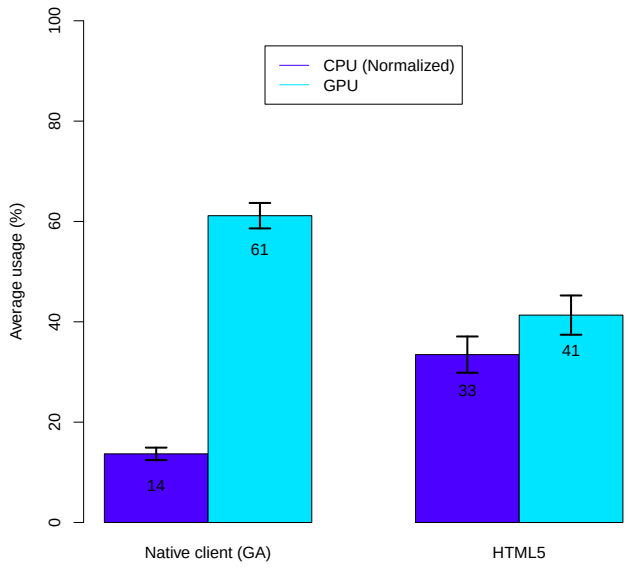

Figure 28: Client side mobile device CPU and GPU usage comparison, Native client (GA) and HTML5 alternative.

7 Discussion

This chapter further analyzes the overall benefits and challenges of the cloudlet model in cloud gaming. A commercial deployment model for the proposed system is also briefly discussed.

7.1 The cloudlet model in cloud gaming

The more distributed model utilizing cloudlets proved to be efficient in decreasing the overall response delay in cloud gaming scenarios. Cloudlets on network edges could enable even the most demanding games to be played remotely. The prototype built is sufficient for a handful of users and could be easily expanded with additional PCs. Even the built prototype could handle at least one additional GPU and therefore another client. The virtual machine model together with the assigned GPUs is an efficient and safe way of dividing the computational resources of a system for multiple users. However the GPU requirement also limits the available existing locations for the cloud gaming servers. Only a handful of service providers currently have GPU-powered instances available on their data centers. This could however change when the prices of GPU-powered instances decrease in the future. Cloud gaming could be a so called killer app for data center instances with GPUs installed.

The availability of cloudlets could also be a problem. It's not feasible to assume that cloudlets could be available everywhere at all times. Thus to make sure the designed cloudlet gaming system is a useful service, there must be a graceful fallback to a distant cloud provider in case a cloudlet is not present. This is the case in the presented design of the system, as a distant cloud server such as the Amazon EC2 would always act as a backup. In the original cloudlet model [30] it was also visioned that the mobile device could itself run the applications if no cloudlets were present. This might not be feasible in the presented scenario since PC games would have to be separately ported for mobile platforms before they could be executed on the device itself.

One of the biggest obstacles in the design when using the cloudlet model is the possibility of huge VM overlays as PC games can be sized anything from a couple of hundred megabytes to several gigabytes. This thesis presented a model where the game images could be separately fetched from a distributed file system. This could be implemented for example by using the existing content delivery network (CDN) infrastructure. Another option would be to store the usable games permanently on the hosts running the cloud servers. New games could be added to the system by uploading the game images to existing cloud locations. This would however restrict the universal approach of the cloudlet model.

The cloudlet model in general has a couple of unanswered weaknesses as well. The base images have to be available on each cloudlet before the VM overlays can be applied. The base images also have to be the same for each cloudlet for the VM overlays to function properly. This might become an issue for example when a security update must be applied to the operating system (base image). After this the base image and the overlays have to be recreated unless the same overlay

strategy is used for updates as well. Nevertheless for the best support the strategy would require a single instance to rule which version of an operating system would be considered the base image.

Overall the universality and the single-hop access to rich resources are the strengths of the cloudlet design. The implementation of the VM synthesis might however be too heavy for software that requires lots of disk space such as games.

7.2 Commercial deployment

The low response delays measured when simulating the game server presence in the premises of the ISP indicate that the ISPs have an advantage in providing low-latency optimized virtual machines for end-users. Currently for example a company called G-Cluster offers operators such cloud gaming solutions and are planning on expanding their services for mobile use as well [5]. As ISPs are all the time looking for new revenue streams this might be something to consider in a larger scale.

The measurements indicate though that a large number of games could be rendered in a distant cloud as well such as the Amazon EC2. By using the more distributed system design presented in this thesis the game service provider could get savings by planning the proximity of the cloud server based on the latency tolerance of the particular game and the quality of the user's network connection. As the mobile network traffic is predicted to rise 11-fold between 2013 and 2018 [2] the ISPs might also want to keep the cloud gaming traffic as local as possible. Thus providing the cloud servers themselves should also save money for ISPs by reducing the traffic going in and out of their networks.

The distributed system could be further expanded by selling small CPU and GPU powered cloud gaming units for home and other local use. The resources of these network-connected units could be utilized also for near-by clients when the unit would otherwise be idle. Using the cloudlet model and the design of the prototype, the sold cloudlet unit could act as a normal PC while sharing resources to other users in the area. As other users would be restricted inside virtual machines, no risk of data breach should be possible. The owner of the cloud gaming unit could receive compensation for example in his/her monthly bill for sharing the computing resources with other users.

The asymmetric nature of home broadband connections could however limit the use of a personal cloudlet as a cloud gaming system outside the local home network. LTE and fiber connections are however gaining ground and could help home users in sharing services such as cloud gaming to their own and others' devices.

7.3 Overall benefits and challenges of offloading computation for mobile devices

The problem statement of this thesis presented six issues that need to be addressed when designing a mobile cloud gaming platform. The first and second problems of strict QoS requirements for the network and available resources can be partly solved

with distributed server locations and by developing the network infrastructure. However as network operators are already tackling with growing network traffic, moving vast majority of gaming to the cloud might inflict too much traffic to the network on top of the already growing video traffic. This might further support the idea of keeping the traffic as local as possible utilizing network operator's own networks and using distant data centers only as backups.

The third and fourth issues regarding the differences in control methods and screen sizes on mobile devices compared to PCs and game consoles can be partly solved with the use of external game controllers and screens. For some games it could be enough just to simulate the keyboard and mouse with the touch screen of the external device. The use of public displays and smart TVs in co-operation with the mobile device could bypass the screen size problem. However because of the different nature of the devices, if could be argued if completely separate games should be designed for mobile devices.

The fifth problem acknowledged the variety of platforms available for mobile devices meaning that a separate remote gaming client would have to be developed for the different platforms. The HTML5 alternative presented in this thesis has the potential of solving this as a web browser is by default installed in all of the platforms. The varying capabilities of browsers at the moment however show that creating such a universal solution is not easy.

The last problem of energy consumption is two-folded. According to results presented in Chapter 6.4, cloud gaming does consume more energy than native single player games. On the other hand native online games can consume even more energy than cloud gaming, as the mobile device needs to execute the game and use the radio for transmission. In this case cloud gaming can actually save energy.

Overall cloud gaming has lots of benefits if these obstacles can be overcome. The possibility to utilize almost unlimited computing resources, achieve platform independence with a cost-effective way is an appealing scenario.

8 Conclusion

Mobile cloud gaming is an emerging new paradigm where computationally weak devices can play games which normally couldn't be run on the devices by using cloud servers to render the game graphics remotely. Its business opportunities have been widely recognized. Its implementation is not trivial though because it has very strict QoS requirements for the underlying network. The overall response delay perceived by the user must be kept as low as possible. Past studies have shown that using distant cloud infrastructure such as the Amazon EC2 is not an optimal solution especially for the most demanding games.

This thesis proposed a new distributed cloud gaming system focusing on the use of cloudlets on network edges. The proposed system is designed to be able to deploy virtual machines for cloud gaming use in different locations based on the delay requirements of the particular game and the network connection of the mobile device. To further increase the QoE of the user, the system is also able to utilize public displays, smart TVs and external controllers.

A prototype Cloudlet Remote Gaming platform was built from open source components to test the benefits of bringing the cloud closer to the user. The prototype used the Elijah cloudlet software and the GamingAnywhere remote gaming software. The Elijah software was used to dynamically deploy the virtual machines. It was modified to support VGA passthrough to be able to assign a GPU for each virtual machine. This way the VMs are capable of running modern PC games. GamingAnywhere (GA) software was used to stream the game video to the mobile devices and to send the control inputs back to server from the mobile device. The GA software was modified to support gamepads both by emulating the mouse and keyboard combination and by creating a virtual controller module capable of creating virtual controllers on the cloud server's OS.

An HTML5 alternative server and client were also developed to support the use of public displays and smart TVs in the proposed system. The benefits of this alternative are that it doesn't require any installations or modifications to existing public display or smart TV platforms. The alternative proved to be efficient on desktop browsers, often used in public display systems, supporting the use of x264 video codec and fragmented MP4 container. In mobile devices the alternative worked only with the VP8 codec with the OGG container that proved not to be fast enough for remote gaming at this point. The support of the gamepad API was also not available yet on mobile devices but worked well with in desktop browsers.

The Cloudlet Remote Gaming prototype was deployed in three distinct locations and the overall response delay was measured. The results were compared to thresholds derived from previous research on the latency requirements of different game types. It was shown that the new distributed model and the use of cloudlets on network edges can benefit the QoE of the user by reducing the response delay significantly. Local cloudlets in the same local network or for instance at the operator's premises can enable even the most demanding fast-paced games to be played remotely.

The power consumption of the mobile device in cloud gaming was also measured.

The results showed that native online games executed on the mobile device could use more energy than cloud gaming. In the case of single player games, cloud gaming consumes more energy because of the constant use of the network. However the power consumption can be significantly lowered by using public displays, Wi-Fi connection and external game controllers.

8.1 Future work

This thesis proposed an outline for a new more distributed cloud gaming system design. Based on the findings of this thesis the development of the system can be continued by implementing the resource provisioning server that chooses the location of the VM deployment. The most optimal way of sharing the game data between cloud servers and cloudlets should be also evaluated in future work. Furthermore a user study should be arranged specifically targeting the use of mobile devices in cloud gaming.

Mobility management was only briefly discussed in this thesis. The ongoing research concerning cloudlets and mobility management could benefit the designed cloud gaming platform.

Overall the response delay measurements could be further adapted to other fields than gaming as well. The original visions of using cloudlets for fields such as augmented reality and speech recognition could also benefit from the found latency measurement results.

References

[1] Business insider: What is a smart tv? http://www.businessinsider.com/what-is-a-smart-tv-2010-12.

[2] Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2013-2018. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.html.

[3] Cloudgaming: Cloud gaming report 2012. http://www.cgconfusa.com/report/.

[4] Elijah cloudlet-based mobile computing. http://elijah.cs.cmu.edu.

[5] G-cluster plans mobile gaming service to challenge OnLive. http://gigaom.com/2012/01/30/g-cluster-plans-mobile-gaming-service-to-challenge-onlive/.

[6] LTE Encyclopedia. https://sites.google.com/site/lteencyclopedia/.

[7] Python-evdev. Evdev documentation. https://pythonhosted.org/evdev/.

[8] Qemu - wikibooks, open books for an open world.

[9] W3C. Gamepad. W3C Working Draft 25 February 2014. http://www.w3.org/TR/2014/WD-gamepad-20140225/.

[10] Niranjan Balasubramanian, Aruna Balasubramanian, and Arun Venkataramani. Energy consumption in mobile phones: a measurement study and implications for network applications. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, IMC '09, pages 280–293, New York, NY, USA, 2009. ACM.

[11] Harry Brignull and Yvonne Rogers. Enticing people to interact with large public displays in public spaces. In *Proceedings of INTERACT*, volume 3, pages 17–24, 2003.

[12] Wei Cai, Victor CM Leung, and Min Chen. Next generation mobile cloud gaming. In *Proceedings of the 7th International Symposium on Service Oriented System Engineering*, SOSE'13, pages 551–560. IEEE, 2013.

[13] Yu-Chun Chang, Po-Han Tseng, Kuan-Ta Chen, and Chin-Laung Lei. Understanding the performance of thin-client gaming. In *Communications Quality and Reliability (CQR), 2011 IEEE International Workshop Technical Committee on*, pages 1–6. IEEE, 2011.

[14] Kuan-Ta Chen, Yu-Chun Chang, Po-Han Tseng, Chun-Ying Huang, and Chin-Laung Lei. Measuring the latency of cloud gaming systems. In *Proceedings of the 19th ACM international conference on Multimedia*, pages 1269–1272. ACM, 2011.

[15] Sharon Choy, Bernard Wong, Gwendal Simon, and Catherine Rosenberg. The brewing storm in cloud gaming: A measurement study on cloud to end-user latency. In *Proceedings of the 11th Annual Workshop on Network and Systems Support for Games*, page 2. IEEE Press, 2012.

[16] Mark Claypool and Kajal Claypool. Latency and player actions in online games. *Communications of the ACM*, 49(11):40–45, 2006.

[17] Preston A. Cox. Mobile cloud computing. *IBM developerWorks*, pages 1–10, 2011.

[18] Hoang T. Dinh, Chonho Lee, Dusit Niyato, and Ping Wang. A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless communications and mobile computing*, 2011.

[19] Kiryong Ha, Padmanabhan Pillai, Wolfgang Richter, Yoshihisa Abe, and Mahadev Satyanarayanan. Just-in-time provisioning for cyber foraging. In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, pages 153–166. ACM, 2013.

[20] Chun-Ying Huang, De-Yu Chen, Cheng-Hsin Hsu, and Kuan-Ta Chen. Gaminganywhere: an open-source cloud gaming testbed. In *Proceedings of the 21st ACM international conference on Multimedia*, pages 827–830. ACM, 2013.

[21] Chun-Ying Huang, Cheng-Hsin Hsu, Yu-Chun Chang, and Kuan-Ta Chen. Gaminganywhere: An open cloud gaming system. In *Proceedings of the 4th ACM Multimedia Systems Conference*, pages 36–47. ACM, 2013.

[22] Junxian Huang, Feng Qian, Alexandre Gerber, Z. Morley Mao, Subhabrata Sen, and Oliver Spatscheck. A close examination of performance and power characteristics of 4g lte networks. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, MobiSys '12, pages 225–238, New York, NY, USA, 2012. ACM.

[23] Michael Jarschel, Daniel Schlosser, Sven Scheuring, and Tobias Hossfeld. An evaluation of qoe in cloud gaming based on subjective tests. In *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2011 Fifth International Conference on*, pages 330–335. IEEE, 2011.

[24] Michael Jarschel, Daniel Schlosser, Sven Scheuring, and Tobias Hossfeld. An evaluation of qoe in cloud gaming based on subjective tests. In *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2011 Fifth International Conference on*, pages 330–335. IEEE, 2011.

[25] Michael Jarschel, Daniel Schlosser, Sven Scheuring, and Tobias Hossfeld. Gaming in the clouds: Qoe and the users' perspective. *Mathematical and Computer Modelling*, 57(11):2883–2894, 2013.

[26] Karthik Kumar and Yung-Hsiang Lu. Cloud computing for mobile users: Can offloading computation save energy? *Computer*, 43(4):51–56, 2010.

[27] Yeng-Ting Lee, Kuan-Ta Chen, Han-I Su, and Chin-Laung Lei. Are all games equally cloud-gaming-friendly? an electromyographic approach. In *Network and Systems Support for Games (NetGames), 2012 11th Annual Workshop on*, pages 1–6. IEEE, 2012.

[28] Tao Lin and Shuhui Wang. Cloudlet-screen computing: a multi-core-based, cloud-computing-oriented, traditional-computing-compatible parallel computing paradigm for the masses. In *Multimedia and Expo, 2009. ICME 2009. IEEE International Conference on*, pages 1805–1808. IEEE, 2009.

[29] Peter Mell and Tim Grance. The nist definition of cloud computing. *National Institute of Standards and Technology*, 53(6):50, 2009.

[30] Mahadev Satyanarayanan, Paramvir Bahl, Ramon Caceres, and Nigel Davies. The case for vm-based cloudlets in mobile computing. *Pervasive Computing, IEEE*, 8(4):14–23, 2009.

[31] Ryan Shea, Jiangchuan Liu, EC-H Ngai, and Yong Cui. Cloud gaming: architecture and performance. *Network, IEEE*, 27(4), 2013.

[32] Ryan Shea, Jiangchuan Liu, EC-H Ngai, and Yong Cui. Cloud gaming: architecture and performance. *Network, IEEE*, 27(4), 2013.

[33] Pieter Simoens, Filip De Turck, Bart Dhoedt, and Piet Demeester. Remote display solutions for mobile cloud computing. *IEEE Internet Computing*, 13(5), 2009.

[34] Omar Soliman, Abdelmounaam Rezgui, Hamdy Soliman, and Najib Manea. *Mobile Cloud Gaming: Issues and Challenges*, pages 121–128. Mobile Web Information Systems. Springer, 2013.

[35] Yu Xiao, Yong Cui, Petri Savolainen, Matti Siekkinen, An Wang, Liu Yang, Antti Yla-Jaaski, and Sasu Tarkoma. Modeling energy consumption of data transmission over wi-fi. *IEEE Transactions on Mobile Computing*, 99(PrePrints):1, 2013.