Stream-A-Game: An Open-Source Mobile Cloud Gaming Platform

Justus Beyer
Quality and Usability Lab
Technische Universität Berlin
Berlin, Germany
justus.beyer@qu.tu-berlin.de

Richard Varbelow
Quality & Usability Lab
Technische Universität Berlin
Berlin, Germany
mail@rivar.be

***Abstract*— Cloud Gaming is a paradigm shift in that games are not executed on the player's device but on a remote server, which delivers a live video stream of the game over the network to the client and takes input commands in return. We present the first truly mobile open-source cloud gaming system, which runs instances of the Android Open Source Project (AOSP) mobile operating system in the cloud and delivers their output in real time to clients running a lightweight display software. Incorporating AOSP allows using this platform to stream innumerable existing Android mobile games which are designed with touch interfaces and mobile usage scenarios in mind. Utilizing a base of mature open-source components, our platform can be configured and manipulated in any conceivable manner.**

Keywords—mobile; cloud gaming; quality; QoE; streaming

I. INTRODUCTION

Since the introduction in 2009 [1], the initial concept of cloud gaming has evolved and is now a commercially available service provided by various providers like SONY, G-cluster or NVIDIA. Typically these providers use proprietary solutions to stream PC, or console games to TVs, PCs or handheld devices. Although these closed-source platforms are of limited value for scientific research, they prove the feasibility of real-time transmission of game content over the Internet, being essential to cloud gaming.

A. GamingAnywhere

In 2013, Chen et al. published the first open-source and freely available cloud gaming platform "GamingAnywhere", capable of streaming applications and games from all major PC operating systems to a PC and an Android client [2]. Hence, GamingAnywhere has been continuously developed as an open-source project and gained a rich set of features such as support for the emerging H.265/HEVC video compression standard. Due to its open-source nature and liberal license, the platform has been used in numerous studies (e.g., [3][4][5]).

B. Android Open Source Project (AOSP)

After its public release in 2008, Android has gradually evolved into the world's most popular smartphone operating system. Although typical Android devices are shipped with considerable amounts of non-free software such as Google's Mail and Play Store apps, the core of the system, the included Linux kernel and an SDK are open-source and freely available in the Android Open Source Project. This has enabled ports of the operating system to countless hardware platforms both by commercial manufacturers as well as by community-driven open-source projects. One of these latter endeavors is Android-x86,[1] which develops an Android branch usable on ordinary PC hardware.

C. Mobile Cloud Gaming

Previous works on streaming games to mobile devices have concentrated on delivering desktop class games to less capable battery-powered handheld devices through the means of cloud gaming (e.g., [6], [7]). This device category change entails the need to adapt the input mechanisms expected by the games (e.g., keyboard, mouse, controller) to means available on the mobile device. While dedicated mobile gaming devices with support for cloud gaming such as SONY Vita or Nvidia Shield offer input options comparable to a console game controller, solutions encompassing general purpose mobile devices such as smartphones or tablets typically employ custom gestures or overlay buttons, which are displayed on-top of the streamed game output. Although these substitute input mechanisms permit bridging the gaps between different device categories, they require the gamer to adapt and may not reach the versatility of the original control they replace.

In contrast to the aforementioned works, the mobile cloud gaming platform we propose does not bridge device category boundaries but streams smartphone and tablet games to those very devices. Our primary aim with this test bed is to facilitate research of the implications of network degradations and delay onto the Quality of Experience (QoE) of mobile cloud gaming with realistic use cases (i.e., with games which are designed and optimized to be played on mobile devices). Additionally, the platform also enables usage scenarios such as interacting with modern resource-demanding apps on outdated mobile devices, easier development and testing of apps without needing to install the tested app locally, and even crossing smartphone operating system boundaries by interacting with Android software on an iOS device.

In the following Section II we briefly outline the structure of the platform. Details about each of the components are given in Section III. In Section IV we evaluate the system's performance. In Section V we outline our demonstration at the NetGames workshop and in the final Section VI we conclude with an outlook on future work.

[1] http://www.android-x86.org

	MacBook Air	Google Nexus 4		Galaxy S4		iPhone 6	
HW video decoder	no	yes	no	yes	no	yes	no
Measured delay	85 ms	235 ms	190 ms	230 ms	169 ms	147 ms	155 ms

Table 1: Measured system feedback delays between user input and visible system response for various devices with and without hardware-accelerated H.264 decoding. The clients connected to the streaming component using an 802.11n WIFI network on an interference-free channel in the 5 GHz band.

II. System Architecture

The platform consists of four distinct building blocks:

- The compute component runs an Android system inside a virtualization environment,
- the rendering component receives OpenGL instructions and textures from the virtual Android and renders them to pixel-based images using a Graphics Processing Unit (GPU),
- the streaming component compresses these rendered images and provides a video stream on the network and
- the client accesses and displays this video stream and transmits input commands back to the server.

These four components compose a pipe in which visual output flows from the virtualized Android to the client and input commands are forwarded vice versa. This modular design allows each component to be independently developed and configured (e.g., the version of the Android system inside the compute component may be altered without implications to the rest of the system).

III. Implementation

The aforementioned design using four components was chosen as it permits reusing existing open-source projects with only moderate modifications.

A. Compute Component

The design goal of making a large number of existing mobile games accessible to cloud gaming QoE research led to choosing AOSP as an execution environment. A large number of games have been developed for Android and the platform's popularity continues to increase. In the proposed platform, the compute unit consists of a Xen virtual machine running a modified installation of Android-x86. This Android distribution has been adapted to work within the Xen virtualization environment. Furthermore, the graphical OpenGL rendering system was adapted to not generate images locally but convert the OpenGL ES instructions to OpenGL and send these rendering commands and textures through a TCP network connection to an external renderer using a process called marshalling. Likewise, marshalled user input is received by a purpose-built service "vinput" and injected into the Android system through virtual keyboard and mouse devices.

B. Rendering Component

To generate pixel-based images, the rendering component executes the OpenGL instructions on a GPU, displaying the generated output on-screen. As this element is implemented as a desktop SDL application, it can be used to interact with the virtual Android device from a PC without the streaming provided by the latter two components.

C. Streaming Component

Based on an extended GamingAnywhere server, the streaming component provides the pixel-based visual output generated by the rendering component as a compressed network stream to connecting clients. For this purpose, the component uses a hooking mechanism to immediately receive newly rendered image buffers as their rendering is completed on the GPU. As a mobile operating system, Android is optimized for low energy consumption, e.g., screen updates are only generated whenever the screen contents change. This requires the deliberate insertion of duplicate frames by this component to output a continuous video stream. Without these injected frames, a newly connecting client would not be able to successfully receive a full image whenever the virtual Android device would show a static screen (e.g., menus or game scenes without animations). Furthermore, visual degradations caused by data corruption or packet loss during transmission to the client (e.g., blockiness, discolorations) would last for extended periods of time, as no keyframe information would be transmitted containing intact image data. Whenever the frame update rate from the rendering component drops, frames are inserted to maintain a rate of at least 80% of the nominal frame rate. The generated output stream therefore has a variable frame rate. During our work on the platform we noticed that the handling of keyframes is critical for both the performance and the visual fidelity of the system: Conventional key frames contain sufficient information to reconstruct a full image of the video stream without referencing previous frames. This inevitably causes spikes in the transmission bitrate of the stream as these full frames require more information to be transmitted than (P-)frames that are allowed to reference image data from previous frames. While this behavior is not problematic for delay-insensitive streams as long as the average bitrate resulting from buffering does not exceed the limits of the transmission channel, games require both low delay (i.e., no buffering) and a constant frame transmission latency (i.e., frame sizes have to be relatively homogenous). In our platform this is achieved through a video coding feature called "Periodic Intra Refresh" (PIR), which omits full keyframes in the video stream and instead gradually delivers reference-free blocks of image data to the client, spreading the overhead to recover one full image over many frames instead of one [8].

D. Client Component

The client component connects to the streaming component and receives, decodes and displays the video stream. In parallel, it registers user input and sends it back over the network. As our streaming component uses a GamingAnywhere server, only minor modifications were necessary to the GamingAnywhere Android client to support

interaction with the virtual Android without on-screen overlay buttons. During a previous study we found that the Android devices under test responded to user input slower than tested iOS devices [9]. Therefore, we chose to also implement a client for the iOS operating system to be able to achieve a lower overall system feedback latency. Like the GamingAnywhere Android client we first implemented both software-based and hardware-accelerated video decoding. We found the latter to perform better both in terms of power consumption and rendering delay, but reacting very sensitive to lost packets and easily irreversibly stalling with PIR-enabled streams. We therefore focused on software-based rendering and further optimized this to lower the delay. In order to reduce load on the device's CPU we moved the color space conversion from the video stream's YUV to RGB from the CPU to the GPU by using shaders to calculate RGB values per pixel.

IV. Performance Evaluation

Using the method described in our previous work [9] we measured the performance of the platform in terms of feedback delay. The tested devices and their respective measured delays are listed in Table 1. For the test we used a PC as server (Intel Core i5-4460 3.2 GHz, 32 GB RAM, AMD Radeon HD 7950) running the XenServer 6.5 SP1 virtualization environment. Two virtual machines were configured on this machine: One executed the compute component (i.e., the virtualized Android), whereas the other (rendering component) ran an Ubuntu Desktop 15.04 64-bit with the physical GPU made available as pass-through device inside the VM. The mobile devices connected to the streaming component through a dedicated 802.11n WIFI network operating on an interference-free 40 MHz channel in the 5 GHz frequency band. During the tests we noted that the different hardware decoders for H.264 cause vastly different play-out delays. We also found, that the touch screens contribute significantly to the overall delay. When we inserted touch events directly within the Xen Android virtual machine (the compute component) through the administration software XenCenter instead of simulating touches on the iPhone 6's touchscreen surface, we registered a drop in feedback delay from to 155 ms to 106 ms (without hardware-accelerated H.264 decoding).

V. Demonstration

During the NetGames workshop, we demonstrate Stream-A-Game by streaming various popular Android games to an iOS device using a downscaled installation of the Stream-A-Game system, in which the server components run on a Laptop with Ubuntu Linux and VirtualBox instead of a server with Xen virtualization.

VI. Conclusion

In this paper we presented Stream-A-Game, a cloud gaming platform to stream Android games to mobile devices. In contrast to other existing cloud gaming platforms, this approach does not entail crossing device category boundaries and therefore requires no modifications to the interaction paradigm inherent to the streamed apps. Besides its intended application as QoE research tool, this platform may also be used to test apps without having to install them locally, accelerating development with the availability of using virtual machine snapshots on the virtual Android device, and interacting with Android application on iOS devices.

To broaden the applicability of the platform, further work is required. The addition of audio in- and output would extend the scope of the system to applications depending mainly on audio. Furthermore, forwarding additional input information from the client to the compute component such as data from gyroscopes or accelerometers and touch pressure would allow apps and games in the virtual Android to behave even more like locally running software.

The source code of the Stream-A-Game platform is published on GitHub: https://github.com/streamagame

Acknowledgment

This work was co-funded by the German BMBF, funding code 01IS12056.

References

[1] P. E. Ross, "Cloud computing's killer app: Gaming," Spectrum, vol. 46, no. 3, pp. 14–14, IEEE, 2009.

[2] C.-Y. Huang, C.-H. Hsu, Y.-C. Chang, and K.-T. Chen, "GamingAnywhere: an open cloud gaming system," in Proceedings of the 4th ACM multimedia systems conference, pp. 36–47, ACM, 2013.

[3] I. Slivar, M. Suznjevic, L. Skorin-Kapov, "The impact of video encoding parameters and game type on QoE for cloud gaming: A case study using the steam platform," in Proceedings of the 7th International Workshop on Quality of Multimedia Experience (QoMEX), IEEE, 2015.

[4] I. Slivar, M. Suznjevic, L. Skorin-Kapov, M. Matijasevic, "Empirical QoE study of in-home streaming of online games," in 13th Annual Workshop on Network and Systems Support for Games (NetGames), IEEE, 2014.

[5] J. Beyer, R. Varbelow, J.-N. Antons, S. Möller, "Using electroencephalography and subjective self-assessment to measure the influence of quality variations in cloud gaming," in Proceedings of the 7th International Workshop on Quality of Multimedia Experience (QoMEX), IEEE, 2015.

[6] K.-T. Chen, Y.-C. Chang, P.-H. Tseng, C.-Y. Huang, and C.-L. Lei, "Measuring the latency of cloud gaming systems," in Proceedings of the 19th ACM international conference on Multimedia (MM '11). pp. 1269-1272, ACM, 2011.

[7] W. Shaoxuan, S. Dey, "Modeling and Characterizing User Experience in a Cloud Server Based Mobile Gaming Approach," in Global Telecommunications Conference 2009, IEEE, 2009.

[8] S. Kumar, L. Xu, M. K. Mandal, S. Panchanathan, "Error resiliency schemes in H.264/AVC standard," Journal of Visual Communication and Image Representation, Volume 17, Issue 2, pp. 425-450, April 2006.

[9] J. Beyer, R. Varbelow, J.-N. Antons, and S. Zander, "A Method For Feedback Delay Measurement Using a Low-cost Arduino Microcontroller," in Proceedings of the 7th International Workshop on Quality of Multimedia Experience (QoMEX), IEEE, 2015.