Bringing Mobile Online Games to Clouds

Dominik Meiländer[1], Frank Glinka, and Sergei Gorlatch
University of Muenster
Muenster, Germany
[1]d.meil@uni-muenster.de

Li Lin, Wei Zhang, and Xiaofei Liao
Huazhong University of Science and Technology
Wuhan, China

***Abstract*—Massively Multi-player Online Games (MMOG) are characterized by intensive interactions between many simultaneous users and real-time demands on Quality of Service (QoS). Other examples of similar, real-time online interactive applications include various virtual environments, as well as multi-pupil e-learning and simulation-based training courses. A highly desirable enhancement for MMOG is the use of mobile devices for accessing the game application. However, the limited computing power of mobile devices is an obstacle for implementing computation-intensive parts of MMOG, in particular graphics processing, on mobile devices. This paper proposes a novel runtime system for mobile MMOG and other similar applications that moves computation-intensive tasks, including graphics processing, from the mobile devices to Cloud resources. We report experimental results of our runtime system using a realistic multi-player online game.**

I. Introduction

This paper is motivated by the challenges of the emerging class of mobile *Massively Multi-player Online Games (MMOG)*. Other popular and market-relevant representatives of similar, real-time online interactive applications include real-time training and e-learning based on high-performance simulation. All these applications are characterized by high performance requirements, such as: short response times to user inputs (about 0.1-1.5 s); frequent state updates (up to 100 Hz); large and frequently changing numbers of users in a single application instance (up to 10^4 simultaneously).

With the rapid advancement of both wireless network technologies and mobile devices, there is an increasing demand for *mobile MMOG* in which users access the game application from their mobile devices. However, a major problem of mobile MMOG is the limited computing power of mobile devices. MMOG processing involves computation-intensive tasks, such as 3D display rendering up to 100 times per second, which cannot be implemented with sufficient speed on mobiles.

Since Cloud computing offers cost-efficient leasing resources on demand, the high performance requirements of mobile MMOG can be satisfied by offloading computation-intensive tasks, like graphics processing, from mobile devices to Cloud resources. However, the bandwidth of contemporary mobile networks is not sufficient for transmitting high-resolution display images up to 100 times per second to a large number of concurrent users. In order to satisfy the high performance requirements of mobile MMOG on Clouds, several challenges have to be addressed: reducing the workload for graphics processing on mobile devices because of their limited computing power; utilizing Cloud resources efficiently for offloading computations from mobile devices; compensating for low bandwidth, high latency and unstable connections in mobile networks.

In this paper, we target these challenges and propose a novel architecture for offloading computation-intensive graphics processing tasks from mobile devices to Cloud resources. Our contribution towards implementing this architecture is a runtime system for mobile MMOG that allows users to access applications from their mobile devices and offloads at least half of the computational overhead required for display rendering to Cloud resources. The main benefit of our approach is the reduced bandwidth consumption due to transmitting small-size 3D instructions as compared to existing solutions that transmit large-size display images [3–5, 7, 8].

We evaluate our runtime system using a realistic online game which has been implemented using our *Real-Time Framework (RTF)* [1] which is a middleware platform for high-level development and efficient execution of online interactive applications that provides efficient mechanisms for distributed processing and monitoring of the application state.

The paper is organized as follows. Section II describes our target class of Massively Multi-player Online Games (MMOG) and the Real-Time Framework (RTF) used for their development and execution. Section III presents our novel architecture for mobile MMOG on Clouds. Section IV reports our experimental results on the execution of a multi-player online game on mobile devices using a prototype implementation of the proposed architecture. Section V compares our approach to related work and concludes the paper.

II. MMOG Development with RTF

The left-hand side of Fig. 1 shows the *real-time loop* model [10] which illustrates the server-side tasks during MMOG execution. Each user is connected to one application server that processes the *user's actions* (e.g., commands and interactions with other users) which are calculated by the application client based on the *user's inputs* on input devices (keyboard, mouse, etc.). One iteration of the real-time loop is called a *tick* and consists of three steps:

1) Each server receives actions from its connected users.
2) Each server computes a new application state according to the received actions and the application logic.
3) Each server sends the newly computed state to its connected users and to other servers.

Steps 1 and 3 involve communication to transmit the users' actions and state updates between multiple application servers. The computation of a new application state (step 2) involves

978-1-4799-3088-3/14/$31.00 ©2014 IEEE

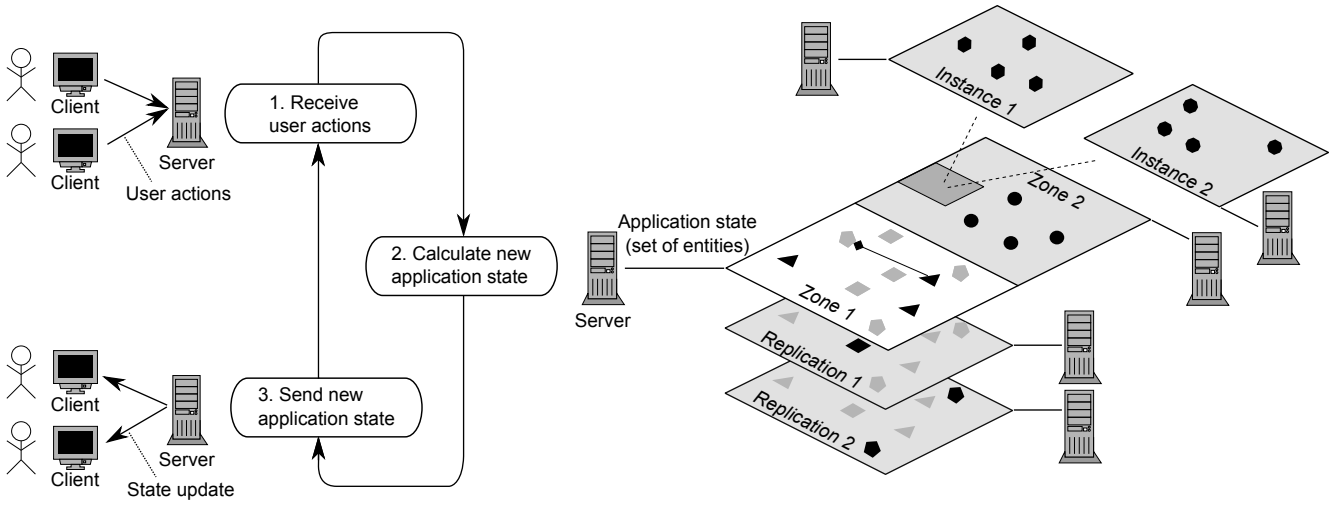

Fig. 1. One iteration of the game real-time loop (left); RTF methods for application state distribution (right).

compute-intensive calculations which apply the application logic to the current state, taking into account the newly received users' actions. Typically, application servers filter state update information by calculating an *area of interest* for each user: only changes that are visible for the user are sent to him, in order to reduce the used network bandwidth. The time required for one iteration of the real-time loop (*tick duration*) is directly related to the application's response time, and, hence, is used as a Quality-of-Experience (QoE) criterion for MMOG.

In our work, we rely on the *Real-Time Framework (RTF)* [10] that has been developed at the University of Muenster as a high-level development platform for MMOG which supports the application developer in three essential tasks as explained in the following: (i) distribution and parallelization of application state computation, (ii) communication handling, and (iii) monitoring and distribution handling. RTF supports three common methods of *application state distribution* among servers for parallelization (on the right-hand side of Fig. 1): zoning, instancing and replication, and combinations of them [10]. RTF provides high-level mechanisms for *communication handling*: automatic (de-)serialization for objects to be transferred over network (user actions, application state updates, etc.), (un-)marshalling of data types, and optimization of the bandwidth usage. RTF's *monitoring and distribution handling* allows the provider to change the distribution of the application state during runtime, as well as receive monitoring data from RTF inside an application server.

III. Utilizing Cloud resources for mobile MMOG

In mobile MMOG, the application software is expected to run on mobile devices. However, the comparatively low CPU and memory capacity, as well as limited battery life of mobile devices, combined with potentially non-reliable, high-latency wireless networks, make it difficult to achieve high QoE, especially for increasing numbers of users.

Cloud Computing opens new opportunities for mobile MMOG to serve very high numbers of users and still comply with QoE demands. This is accomplished by leasing Cloud resources on demand and offloading computations from mobile devices to more powerful Cloud resources. Despite a variable number of users, Cloud resources can be used efficiently if the application supports adding/removing resources during runtime.

The left-hand side of Fig. 2 shows a common modern system architecture for MMOG and illustrates the client-side tasks during MMOG execution, as supported by RTF. Distributed application processing is implemented in a continuous loop which consists of four major steps:

1) The clients compute user actions (based on the users' inputs) and send them to the application server.
2) The server computes a new application state according to the received actions from all clients and the application logic.
3) The clients receive the new application state from the server.
4) The clients render new display images to present the new application state to the users.

For mobile MMOG, we propose a new architecture; we introduce an additional system component called *proxy client* that acts as an intermediary between the mobile device and the MMOG server. For each mobile device, a dedicated proxy client is started on Cloud resources which are realized as virtual machines (as usually in Cloud Computing). The proxy client is used for offloading the computation-intensive calculation of rendering instructions from the mobile device.

The right-hand side of Fig. 2 illustrates how our envisioned architecture implements the MMOG execution in six steps:

1) The mobile device captures and sends user inputs to the proxy client which is running in the Cloud.
2) The proxy client computes corresponding user actions and sends them to the application server (potentially also running in the Cloud).
3) The server computes a new application state according to the application logic.
4) The proxy client receives a new application state from the server.
5) The proxy client calculates rendering instructions reflecting the new application state, and transmits them to the mobile device.

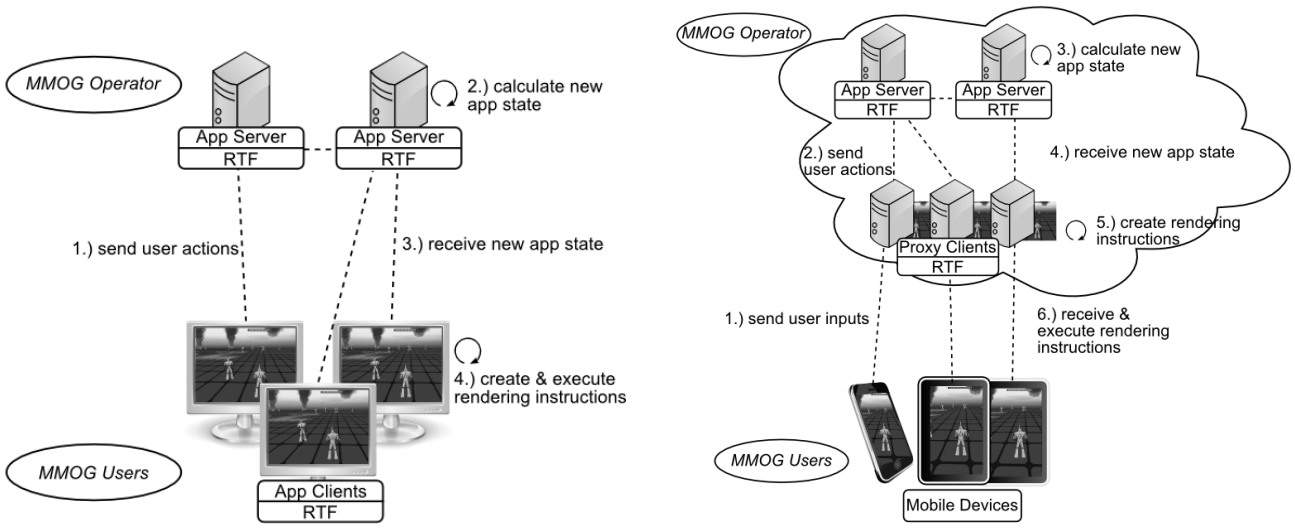

Fig. 2. Traditional (left) and mobile (right) system architectures for MMOG.

6) The mobile device receives and executes the rendering instructions for display rendering.

The proxy client needs to address the three main tasks that are executed by the application client in the traditional system architecture for MMOG (left-hand side of Fig. 2):

(a) sending user actions to the application server (step 1),

(b) receiving the latest application state from the application server (step 3 in the figure),

(c) creating and executing rendering instructions for display rendering (step 4).

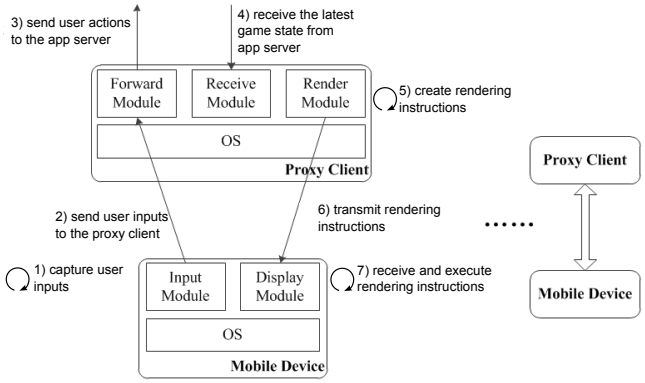

Fig. 3. Proxy client's architecture and its communication with mobile device.

Fig. 3 shows that, according to the three tasks (a)-(c) described above, the proxy client consists of three main modules (each module addressing one task, see Fig. 3):

- the *forward module* sends the user's operations to the application server (task (a)),
- the *receive module* gets the latest application state from the application server (task (b)),
- the *render module* creates rendering instructions and transmits them to the mobile device (task (c)).

The render module plays the major role: it applies the application logic to the received application state. While rendering instructions are typically executed right after their creation, the render module intercepts the execution of rendering instructions and transmits them to the mobile device for local execution. Thus, the proxy client offloads computation-intensive tasks from the mobile device to Cloud resources, such that the mobile device only needs to execute the received rendering instructions for display rendering. By decoupling the creation of rendering instructions from their execution, mobile MMOG can be executed efficiently on mobile devices. For typical MMOG, the creation of 3D instructions requires at least 50 % of the display rendering time and hence the mobile device offloads at least half of its computational load to the Cloud.

On the mobile device, we implement two major communication modules for execution on the mobile device: (i) the *input module* captures the user's input operations, and sends them to the proxy client, and (ii) the *display module* receives and executes the rendering instructions from the proxy client. The input module utilizes the SDL input library [6] which is a cross-platform media library that provides low-level access to keyboard, mouse and other input devices and handles the user inputs on the mobile device. SDL captures user inputs and sends them to the proxy client.

In order to target the specific challenges caused by the limitations of the mobile network (limited bandwidth, unstable connections, etc.), we implement the following three optimizations for efficient communication between the proxy client and the mobile device:

- A *caching mechanism for buffering reusable rendering instructions*. Both the proxy client and the mobile device implement an instruction buffer. The render module checks its buffer state before transmitting rendering instructions, such that only updated instructions are transmitted to the mobile device. Then, the mobile device combines its buffered instructions and the newly received instructions for appropriate display

rendering. According to our analysis, the caching mechanism can reduce network traffic by up to 50 %.

- *Compression of rendering instructions and compensation of packet loss*. Graphics approximations like the A2BGR10 format [14] are adopted for the compression of vertex data (i.e., data for representing position, color, etc. of 3D objects). In order to address the unreliability of mobile networks, the render module can compensate for packet loss while keeping the display consistent. For this purpose, rendering instructions are categorized into two categories: 3D instruction data (for transformation of 3D objects and applying special effects, etc.) and geometry data (containing index and vertex data). In its current implementation, the loss of 3D instruction data can be compensated to a certain degree, but geometry data must be kept consistent.
- *Adaptive graphics level of detail (LOD)*. The representation of 3D objects is defined for different quality levels. If the network provides high bandwidth and stable connections, the proxy client chooses high quality representations of 3D objects; low-quality representations are rather chosen for low-bandwidth networks.

Another challenge is the efficient utilization of high-performance Cloud resources which may, e.g., include multiple GPUs (Graphics Processing Units). Our proposed system architecture utilizes the Xen hypervisor [9] for hardware virtualization that allows to run multiple proxy clients on the same resource (Fig. 4). For this purpose, each proxy client is executed in a dedicated VM (called Dom1, Dom2, etc.). Multiple VMs (each running one proxy client) are executed in parallel on the same resource and managed by the virtual machine monitor (called Dom0).

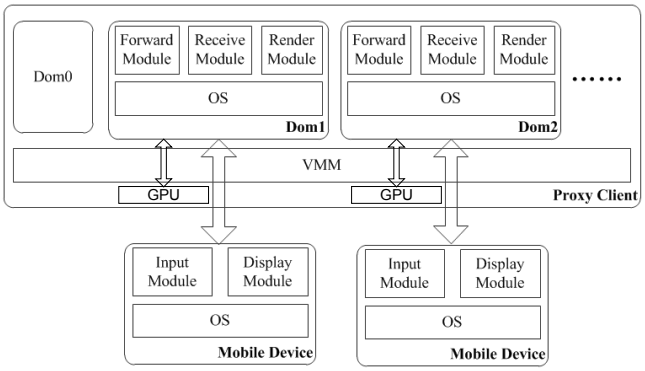

Fig. 4. Execution of multiple proxy clients on a multi-GPU resource.

However, GPU virtualization is not supported by the Xen hypervisor and although hypervisors with support for GPU virtualization exist, they generally show a poor performance which cannot satisfy the high performance requirements of mobile MMOG. Hence, each proxy client needs to access one GPU exclusively which results in a limited number of proxy clients that can be executed in parallel on the same resource. Therefore, our proposed system architecture provides a dedicated GPU to each proxy client and, hence, utilizes multi-GPU resources by assigning each GPU exclusively to one proxy client running inside a VM. While the forward module und the receive module only induce small workload on the CPU, the requirements of multiple render modules cannot be satisfied by a single GPU which cannot ensure the performance of multiple MMOG instances concurrently due to the high workload required for creating rendering instructions; moreover, most rendering engines work in exclusive working mode. For this purpose, our system has a one-to-one correspondence between GPUs and proxy clients.

As a personal terminal, every mobile device is meant to support only one user. For this purpose, each mobile device has one input module and one display module. The input module is connected to the forward module of one particular proxy client and the display module is connected to the corresponding render module. Since each mobile device is connected to one proxy and each proxy is only capable of creating 3D instruction for one user (due to having access to a single GPU), proxy clients and mobile devices have a one-to-one correspondence; i.e., for each mobile device, a dedicated proxy client has to be started on a virtual machine (VM) in the Cloud.

IV. Evaluation

As the application example for our experiments, we use the RTFDemo application which is a fast-paced action game from the domain of FPS (First-Person Shooter), with all typical features of modern online games (Fig. 5).

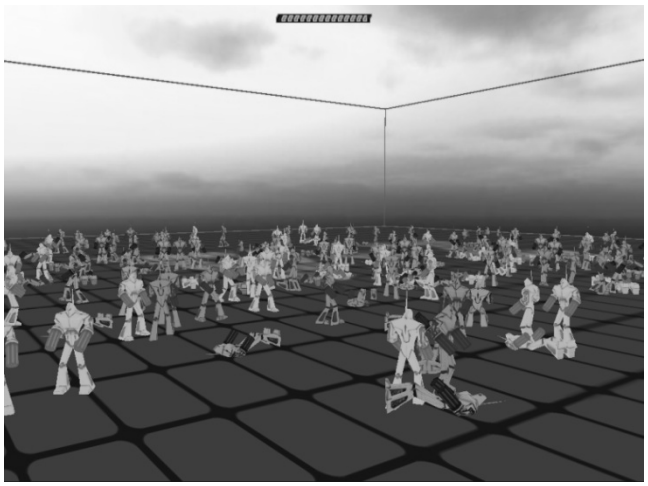

Fig. 5. A screenshot of the RTFDemo application.

In RTFDemo, each user controls his own avatar (robot) in the 3D virtual world, and users can interact by shooting at and thus damaging other users' avatars. Users are simulated by so-called bot clients which are dedicated client processes that autonomously trigger movement and attack actions. To resemble the strong performance demands of mobile online games, we set the number of targeted updates of the game state (which is a parameter of the RTFDemo application) to 25 updates per second, which is currently viewed as a comparatively high update rate for FPS games. In order to provide a seamless gaming experience, the response time is required to be below 120 ms on average.

Our setup for evaluating the architecture that integrates RTF and the proxy library is as follows. Two RTFDemo server instances run on an Intel Pentium E5300 2x2.6 GHz, 2 GB RAM, Intel G41 Express Chipset, and are responsible

for simulating the virtual environment; they replicate a single zone. On the proxy client, an Intel Xeon E5620 4x2.4 GHz, 12 GB RAM, Asus Radeon HD 6990, two VMs are responsible for running two graphical RTFDemo clients on behalf of two connected mobile devices, i.e., one graphical client for each device. The mobile devices are represented by two laptops running Windows 7 or 8, being an Intel Core 2 Duo P8700 2x2.53 GHz, 4 GB RAM, ATI Radeon HD Graphics 3400 for Mobile Device 1 and an Intel Core i7-3517U 2x1.90 GHz, 4 GB RAM, Intel HD Graphics 4000 for Mobile Device 2.

During each of our experiments, increasing numbers of computer-controlled clients (bots) are connected to the two servers in multiple waves of 20 additional clients per server. Adding more clients to the game increases the processing, network and 3D scene complexity, thus stressing the system.

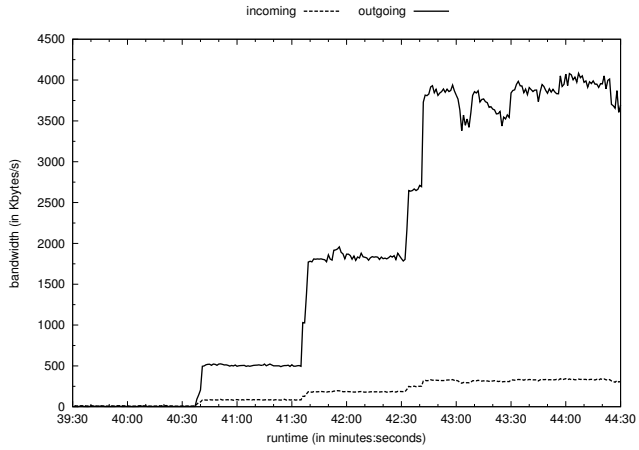

Fig. 6. Incoming and outgoing traffic at the application server.

Fig. 6 shows the incoming and outgoing network traffic on the application server. This includes the actions sent to the server by the VMs on the proxy client and by the bots, as well as the outgoing replication traffic of the game state to the bots and to the VMs on the proxy client which use this information to render the scene. The bandwidth rises for the three first waves of increasing bot numbers, changing the number of clientss after each minute from 0 over 20, 40 and so on, until 80 clients, thus leading to an overall number of four waves and the overall duration of 5 minutes per experiment (experiment is continued for one more minute after the last wave). However, we observe that the addition of 20 more clients from 60 to 80 does not raise the load any further. The experiment was run in a very large campus network with a high amount of background traffic. We assume that this traffic limited the number of transmissions that could be handled, thus leading to a cap of network traffic for 80 clients. Furthermore, the resulting 3D scene for 80 clients is very complex and dynamic due to the high interactivity between the clients.

For the following experiments, we denote the virtual machines that are executed on the proxy client for each mobile device by Proxy Client 1 and Proxy Client 2, correspondingly; both VMs are executed on one physical resource. Fig. 7 shows the network traffic of the virtual machines Proxy Client 1 and Proxy Client 2. Each of them has to process incoming traffic for receiving state updates from the application servers, as well as incoming traffic for receiving inputs from the connected

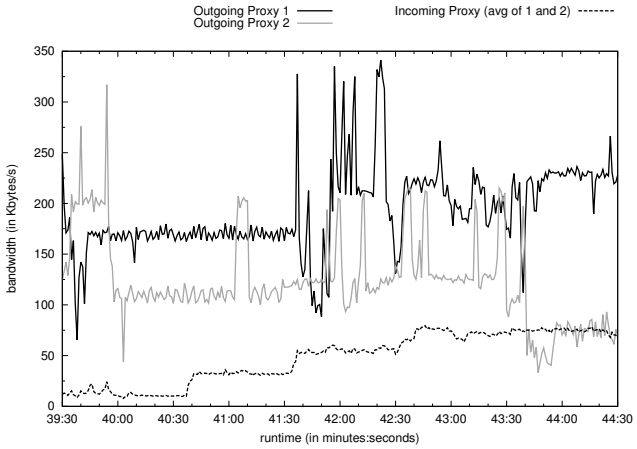

Fig. 7. Network traffic at the proxy client.

mobile device (the latter traffic is neglectable because its amount is independent of the overall number of clients, and the overall amount of input traffic is rather low as compared to replication traffic). Since the incoming traffic of Proxy Client 1 and Proxy Client 2 is very similar, Fig. 7 shows the average incoming traffic as a single curve. Outgoing traffic includes the 3D instructions sent from each proxy client to its corresponding mobile device, and the resulting client actions of the input processing for the server (the latter is low and independent of the overall number of clients, and the overall amount of traffic for action forwarding is rather low as compared to the traffic for sending 3D instructions). Apparently, the outgoing traffic is very variable depending on the viewport selected by the user at the mobile device: if the user, e.g., looks into the sky, then fewer 3D instructions for all the avatars have to be sent to the mobile device as they are not part of the rendered scene. Therefore, the network load is highly dependent on the actions of the clients and hard to predict.

Fig. 8 shows the corresponding mobile device related traffic. Since the outgoing traffic for sending client actions is rather neglectable, we omit it in the figure. As expected, the incoming traffic for rendering instructions (shown in Fig. 8) corresponds to the outgoing traffic of the proxy clients (Fig. 7).

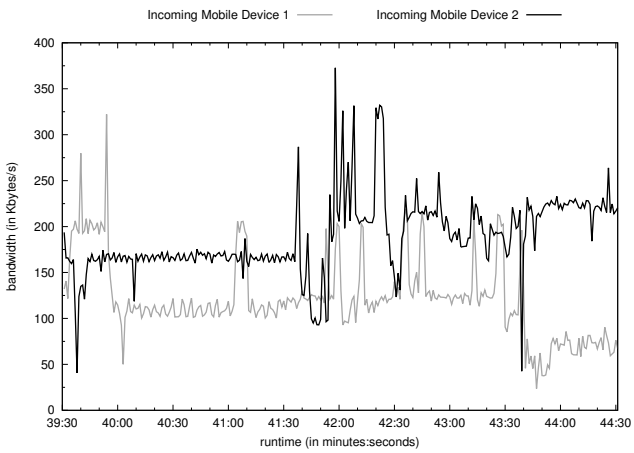

Fig. 8. Network traffic at the mobile devices.

Finally, we measured the response time between the application server and the proxy clients, i.e., the time between receiving a user input on the proxy client and receiving the corresponding game state update from the server; this time resembles the traditional response time property of dedicated clients which are directly connected to the server. Moreover, we measured the response time between the mobile device and its proxy client which consists of the sum of two times: 1) the time between capturing a user input on the mobile device and receiving the input on the proxy client, and 2) the time between receiving a game state update on the proxy client and receiving the rendering instructions on the mobile device. The overall response time is then the sum of the response time between server and proxy client and the response time between mobile device and proxy client.

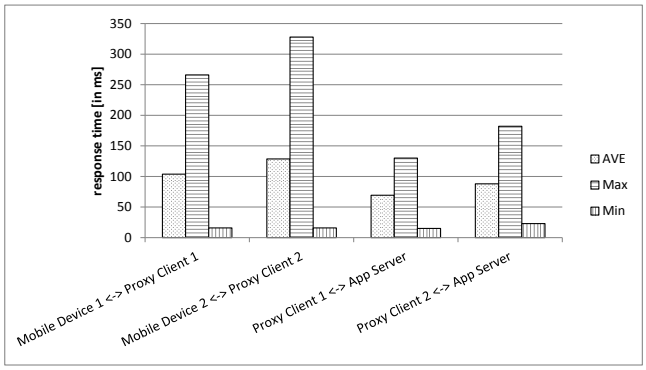

Fig. 9. Response time between the involved components.

Fig. 9 shows that the average response time between the mobile device and its proxy client is higher than the average response time between the proxy clients and the application server. While the minimum response time is equally low in all cases, the maximum response time is considerably higher for the transmission of rendering instructions as compared to receiving game state updates. These results show the potential for optimizing the location of the proxy client in large-scale networks (to minimize the latency between the proxy client and its mobile device), as well as the demand for optimizing the implementation of rendering instruction transmission.

V. Conclusion and Related Work

This paper presents a novel runtime system for mobile Massively Multi-player Online Games (MMOG) that utilizes proxy clients for offloading computation-intensive tasks from mobile devices to Cloud resources. We have shown how the implementation of the proxy client addresses the specific challenges of mobile Cloud computing: 1) reducing the workload for graphics processing on mobile devices, 2) utilizing Cloud resources efficiently, and 3) compensating for low bandwidth, high latency and unstable connections in mobile networks. We have demonstrated the practical relevance and efficieny of our runtime system in a real-world case study using a dynamic multi-player online game. Our experiments illustrate how a particular MMOG is executed on mobile devices and demonstrate the efficiency of our proposed runtime system.

The challenging domain of mobile gaming has been targeted by many researchers and industrial companies. Several commercial [2–5, 7, 8] and open-source [11] solutions are available which have in common that they use Cloud resources for computing display images that are then compressed and transmitted over the network as a video stream. In contrast, our proxy client computes rendering instructions in the Cloud and sends them over the network for execution on the mobile device, thus reducing the consumed network bandwidth and making it more suitable for very frequently updated MMOG. In [13], the authors present an online gaming platform that computes display images in different resolutions according to the user's device, e.g., 1600x1440 for PCs and laptops, 640x480 for smartphones. While our approach transmits rendering instructions, rather than display images, the proxy client implements a similar approach that adapts the quality of the 3D objects to the constraints of the network and mobile device. The European project Games@Large [12] implemented a remote gaming architecture and runtime system executing video games on a central high-performance resource which computes rendering instructions and transmits them to end devices (TV, mobile devices, etc.) for execution. While the Games@Large project and our approach have in common that they offload the computation of rendering instructions from end devices, our approach targets online games with large numbers of concurrent users and provides several novel contributions including different optimizations (caching and compression of rendering instructions and an adaptive level of detail) and the use of Cloud Computing.

Achnowledgment: We are grateful to the anonymous reviewers for their helpful remarks on the preliminary version of the paper. Our research has received funding from the EC's 7th Framework Programme under grant agreements 295222 (MONICA) and 318665 (OFERTIE).

References

[1] "Real-Time Framework (RTF)," http://www.real-time-framework.com.

[2] "Blaast," http://www.blaast.com, 2014.

[3] "G-cluster Global," http://www.g-cluster.com, 2014.

[4] "Gaikai," http://www.gaikai.com, 2014.

[5] "OnLive," http://www.onlive.com, 2014.

[6] "Simple DirectMedia Layer (SDL)," http://www.libsdl.org, 2014.

[7] "StreamMyGame," http://www.streammygame.com, 2014.

[8] "t5 labs," http://www.t5labs.com/, 2014.

[9] "The Xen Project," http://www.xen.org, 2014.

[10] F. Glinka, A. Ploss, S. Gorlatch, and J. Müller-Iden, "High-Level Development of Multiserver Online Games," *International Journal of Computer Games Technology*, vol. 2008, no. 5, pp. 1–16, 2008.

[11] C.-Y. Huang, C.-H. Hsu, Y.-C. Chang, and K.-T. Chen, "Gaminganywhere: an open cloud gaming system," in *Proc. of the 4th ACM Multimedia Systems Conf.*, ser. MMSys '13. ACM, 2013, pp. 36–47.

[12] A. Jurgelionis, P. Fechteler, P. Eisert *et al.*, "Platform for Distributed 3D Gaming," *International Journal of Computer Games Technology*, vol. 2009, pp. 1:1–1:15, 2009.

[13] S.-S. Kim and C. Cho, "Multiscreen-based Gaming Services using Multi-view Rendering with Different Resolutions," in *Proc. of the 2nd International Conference on Mobile Services, Resources, and Users (MOBILITY 2012)*.

[14] E. Persson and Avalanche Studios, "Creating vast game worlds: Experiences from Avalanche Studios," in *ACM SIGGRAPH 2012 Talks*, ser. SIGGRAPH '12. ACM, 2012, pp. 32:1–32:1.