RTF+Shark: Using Software-Defined Networks for Multiplayer Online Games

Tim Humernbrum*, Stefan Delker†, Frank Glinka*, Folker Schamel†, Sergei Gorlatch*
*University of Münster, Germany, {glinkaf, gorlatch, humernbrum}@uni-muenster.de
†Spinor GmbH, Germany, {stefan.delker, fmschamel}@spinor.com

***Abstract*—We demonstrate how Massively Multiplayer Online Games (MMOG) can benefit from the emerging Software-Defined Networking (SDN) technology. We integrate Shark 3D^{TM} – an industry-strength 3D game development framework – with the Real-Time Framework (RTF) – an open-source C++ library enhanced for SDN that provides development and runtime support for MMOG. This integration enables game developers to exploit SDN for guaranteeing network QoS in MMOG. A case study of an MMOG-like application in a real, SDN-enabled network shows the advantages of our approach.**

I. Introduction

Massively Multiplayer Online Games (MMOG) connect a potentially very high number of players who interact with the game world and with each other in real time; therefore, MMOG make strong QoS requirements both on the computational resources and on the underlying network. Moreover, these requirements are dynamic, i. e., they may vary at runtime. This makes MMOG development a challenging task and requires the developer to have detailed knowledge of game design and networking. To facilitate game development, systems such as id Tech and Big World support the developer by providing gaming-typical aspects, e. g., graphics, physics, AI, etc., but the important features of networking and QoS management are often missing.

Our contribution is the integration of Shark 3D^{TM} [1] – a commercial, real-time middleware and 3D engine – with the Real-Time Framework (RTF) [2] – a research-motivated, open-source C++ library for development and runtime support of MMOG including network QoS management. This *RTF/Shark Integration* (called *RSI* throughout the paper) allows the game developer to specify QoS requirements which are then enforced in the network by exploiting the emerging Software-Defined Networking (SDN) technology: a game application requests the *SDN controller* at runtime to accommodate certain QoS requirements for specific users; the controller then enforces the requested QoS, e. g., by redirecting or prioritizing the traffic of these users.

Recent approaches [3-5] to managing QoS using SDN aim at a generic QoS functionality and, therefore, are hard to integrate in real-time applications like MMOG because they require the developer to map low-level networking details to high-level game demands. In contrast, our RSI approach adapts SDN-based network QoS management to the typical structure of MMOG in a commercially successful framework (Shark) and provides the developer with high-level QoS metrics which ease the game development process.

II. QoS for MMOG with RTF and Shark

In a typical MMOG, user inputs are processed by the game clients and sent as *actions* to the game server. The server applies the received actions to the game state according to the game logic and sends corresponding *state updates* back to the clients. This happens in an infinite loop, called *real-time loop*. The game state is often distributed among multiple servers to improve the performance and scalability of MMOG.

When designing a particular MMOG, the game developer deals with several tasks regarding the network and its QoS perceived by the players. For instance, the developer has to organize the efficient transfer of data structures that realize actions and state updates. If the game is distributed over several servers, the distributed state computation and the necessary communications for synchronizing the game state across the servers also have to be managed. Moreover, the communication between a server and the clients or other servers usually comprises several data flows with different requirements on network QoS which must be taken into account.

The Real-Time Framework (RTF) [2] provides high-level development support for MMOG. RTF includes the so-called *SDN Module* [6] that extends RTF's high-level interface for the specification of QoS requirements on specific data flows. QoS requirements in RTF consist of one or several so-called *metrics*, together with a value for each metric to be complied with. Traditional metrics in network QoS are, e. g., throughput, packet loss, and jitter. We call them *network-level metrics*, because they are tightly coupled with the underlying network and can be measured and influenced in the network without any knowledge about the application. However, from the game developer's point of view, network-level metrics are too low-level and not directly related to *application-level metrics* like, e. g., the response time. Consequently, developers tend to specify stricter QoS requirements than necessary and to not release reserved network capacity when possible.

To address this problem, the RTF with its SDN Module allows the game developer to specify application-level metrics which are then translated at runtime into network-level metrics understood by the SDN controller, see [7]. In this translation process, application monitoring data provided by the RTF are used. If the application demands change, the network-level QoS requirement is automatically updated by the SDN Module and transferred to the controller.

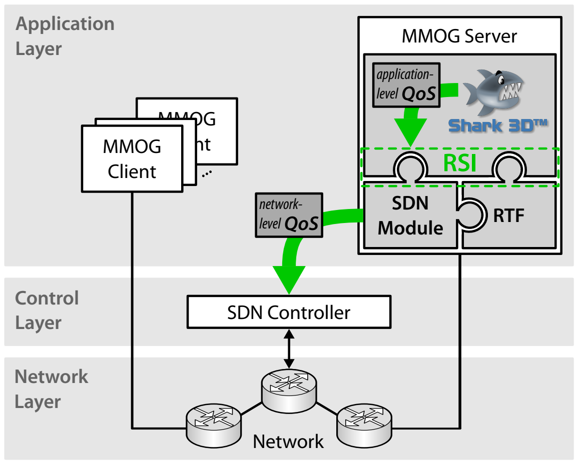

Fig. 1. Basic architecture of the RTF+Shark integration.

Figure 1 depicts the basic architecture of the RSI and shows how Shark is connected with RTF and its SDN Module. Due to the integration, Shark becomes able to use the generic, high-level entity-based networking interface of RTF as a unified API for networking, instead of low-level TCP/UDP connections. It also benefits from the SDN features of RTF: via the C++ API of the SDN Module, MMOG developers can specify QoS requirements based on network- and application-level metrics for particular data flows. At the same time, Shark provides RTF with additional high-level information and runtime monitoring data about game entities and their relationships in the 3D world, which RTF uses to improve the translation of application-level into network-level QoS requirements, predict future QoS demand, etc. RSI also allows the game developer to dynamically accommodate QoS requirements depending on the game state. For instance, a minimum QoS in a specific area of the virtual world, which is expected to be visited by a huge amount of players simultaneously, may only be guaranteed for a premium user paying subscription fee.

The original Shark does not have an entity concept, but rather synchronizes the complete game state between server and client. Hence, the most challenging aspect in the design of RSI is to adapt Shark to RTF's concept of using entities. Our approach is twofold: a) we reuse components of Shark as far as possible, in order to maintain the internal details of how individual engine components like animation controllers are implemented and the protocol they use to communicate with each other; b) we replace the networking components of Shark by new, RTF-based components. As a result, RTF controls all steps of the MMOG's real-time loop, in particular the communication between servers and clients. Shark remains responsible for simulating and rendering the virtual 3D world, as well as for serializing internal components of each entity.

The RSI is implemented on two layers: in the Shark engine and in the authoring editor. In the engine, we have implemented a management mechanism for connecting each RTF entity with its own so-called netlink responsible for the communication with all engine components belonging to that RTF entity. In the Shark authoring editor, we expose the entity concept to the MMOG developer. Without RSI, the editor is used in the following way to create elements of the 3D world, e. g., characters: multiple so-called *prefabs* are used to define different character types in a property tree. The child nodes of these prefabs, which are defined by the developer in the authoring editor, specify the features of the characters like their animation system, physics, and logic scripts. After defining the character prefabs, they can be instantiated by the developer in so-called instance nodes which represent the actual character in the virtual world. With RSI, we introduce a new instance node type in Shark that defines which prefabs build an entity in the RTF sense. In this way, the developer provides the information required by RTF about how the overall 3D world is structured. While this looks like extra work for the MMOG developer in comparison to development without RSI, this will usually make no difference, because, in any case, typical Shark users use prefabs to structure a 3D world into entity-like components. Hence, existing Shark projects can be adapted quite easily to use RSI.

III. Case Study: Online Virtual Production

We perform our evaluation case study in the FIBRE testbed which comprises OpenFlow islands in Europe and Brazil. In our experiments, we used exclusively the Barcelona island of FIBRE whose topology is depicted in Figure 2. The island consists of three NEC IP8800/S3640-24T2XW OpenFlow-enabled switches (*s1*–*s3* in the figure) which connect the SuperMicro SYS-6010T-T server machines (*h1*–*h4*) at 1 Gbit/s. As the servers lack dedicated, DirectX 11-compatible graphics cards needed in our experiments, we added two HP 8760w mobile workstations (*e1* and *e2*) to the testbed.

Our test application is a client/server-based Online Virtual Production (OVP) editor developed with Shark. Using this editor, movies can be created in a virtual 3D world in real time. OVP is a real-world, industry-relevant application featuring MMOG characteristics (e. g., a large number of users which interact with each other in real time). In our experiments, the OVP server application is executed on workstation *e1*, and the OVP clients are executed on *e2*.

In order to experimentally evaluate the effect of managing QoS in SDN-enabled networks using RSI, we distinguish two types of OVP clients: premium clients (for which QoS requirements are managed) and normal clients (no QoS management). With these two types of OVP clients, we demonstrate in two

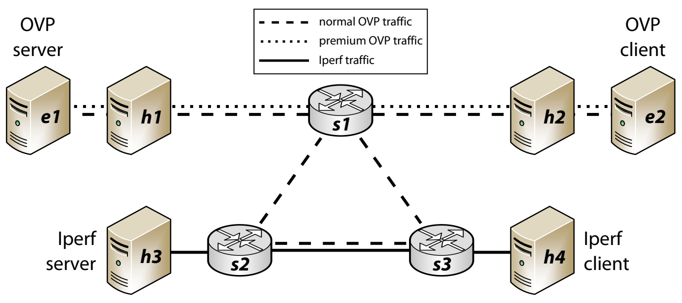

Fig. 2. Experiment topology in the Barcelona island of the FIBRE testbed.

separate experiments how the network-level metric *minimum throughput* and the application-level metric *maximum response time* can be utilized by the developer to specify and enforce QoS requirements of OVP.

In our experiments, the network is controlled by the NCL SDN controller [8] which initially configures the network, such that both normal and premium OVP traffic between *e1* and *e2* is forwarded via switches *s1*, *s2*, and *s3*. During each experiment, we generate background traffic on the route between *s2* and *s3*, in order to simulate a congested route which may lead to a violation of the requested QoS. To fulfill QoS requirements, the SDN controller will reroute premium OVP traffic directly via *s1* (i. e., without the detour via *s2* and *s3*) as depicted in Figure 2.

Figure 3 shows the results of measuring the response time as the main indicator for the QoS perceived by the end-user. In OVP, a response time of up to 100 ms is considered satisfactory because this delay cannot be noticed by the human user. We also measure the packet size for each OVP client. Figure 3 a) shows the average response time for 5 clients with no QoS management, i. e., no QoS requirements are specified. In the time interval from second 20 to 140, we create congestion in the network between *s2* and *s3*. As expected, the clients suffer from the congestion: the average response time increases above 100 ms. The average packet size is approximately the same during the complete experiment execution.

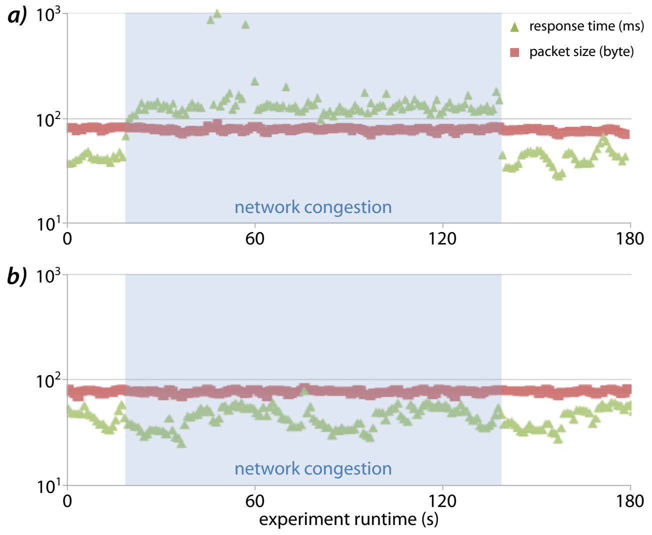

Fig. 3. First experiment: a) no QoS management, b) QoS management enabled

In Figure 3 b), we use RSI to specify a QoS requirement with a minimum throughput of 10 kb/s at the beginning of the experiment for each OVP client. We observe that the response time remains below 100 ms: the congestion interval between second 20 and 140 has no adverse effect on the measured response time because the SDN controller accommodates the requested QoS by rerouting the clients' traffic directly via *s1*.

Our second experiment evaluates the QoS management using response time as the application-level metric. Moreover, the application-level QoS requirements for premium clients

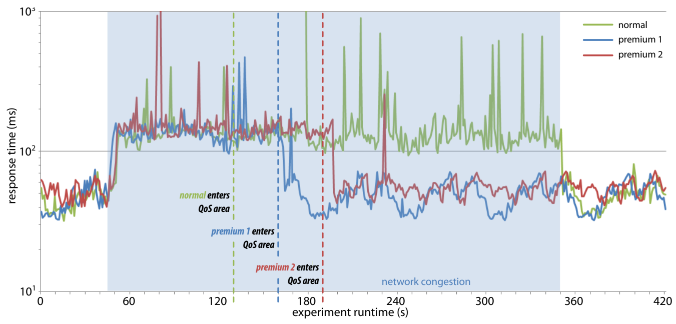

Fig. 4. Second experiment: response time of normal and premium clients.

are not sent to the SDN controller immediately, but rather triggered when a client's avatar enters a specific "QoS area" of the virtual world. This area is defined at development time to demonstrate the RSI feature of dynamic QoS requirement accommodation depending on the player's position. In this experiment, one normal and two premium OVP clients enter the QoS area successively and stay there.

Figure 4 shows the results of the second experiment. Similarly to the first experiment, we create a network congestion between *s2* and *s3* in the time interval from second 45 to 350, which leads to the measured response time for all clients rising above 100 ms. At second 130, the normal OVP client enters the QoS area, but its response time remains above 100 ms because no QoS requirement is triggered. When premium client 1 enters this area at second 160, the QoS requirement for a maximum response time of 100 ms is sent to the SDN controller. Subsequently, the response time of premium client 1 falls below 100 ms. The same effect can be observed when premium client 2 enters that area at second 190. Summarizing, while the normal client suffers from the network congestion, the requested maximum response time for premium clients can be enforced in the SDN network,

References

[1] Spinor GmbH, "Shark 3D Overview & Features," `http://www.spinor.com/features.html`, 2015.

[2] F. Glinka, A. Ploss, S. Gorlatch *et al.*, "High-level Development of Multiserver Online Games," *International Journal of Computer Games Technology*, vol. 2008, pp. 3:1–3:16, Jan. 2008.

[3] A. D. Ferguson, A. Guha, C. Liang *et al.*, "Participatory networking: an API for application control of SDNs," in *Proceedings of the ACM SIGCOMM 2013*. ACM, 2013, pp. 327–338.

[4] M. Bari, S. Chowdhury, R. Ahmed *et al.*, "PolicyCop: An Autonomic QoS Policy Enforcement Framework for Software Defined Networks," in *IEEE SDN for Future Networks and Services (SDN4FNS)*, Nov 2013.

[5] S. Huang and J. Griffioen, "HyperNet games: Leveraging SDN networks to improve multiplayer online games," in *18th International Conference on Computer Games: AI, Animation, Mobile, Interactive Multimedia, Educational Serious Games (CGAMES'13)*, 2013, pp. 74–78.

[6] S. Gorlatch, T. Humernbrum, and F. Glinka, "Improving QoS in Real-Time Internet Applications: From Best-Effort to Software-Defined Networks," in *International Conference on Computing, Networking and Communications 2014 (ICNC)*, Feb 2014, pp. 189–193.

[7] S. Gorlatch and T. Humernbrum, "Enabling High-Level QoS Metrics for Interactive Online Aapplications Using SDN," in *International Conference on Computing, Networking and Communications 2015 (ICNC)*, Feb 2015, pp. 707–711.

[8] I. Bueno, J. Aznar, E. Escalona *et al.*, "An OpenNaaS Based SDN Framework for Dynamic QoS Control," in *IEEE SDN for Future Networks and Services (SDN4FNS)*, Nov 2013, pp. 1–7.