# Dissecting the protocol and network traffic of the OnLive cloud gaming platform

**Article** *in* Multimedia Systems · March 2014

DOI: 10.1007/s00530-014-0370-4

Marc Manzano
Technology Innovation Institute (TII)
**19** PUBLICATIONS **67** CITATIONS
SEE PROFILE

Manuel Urueña
University Carlos III de Madrid
**65** PUBLICATIONS **328** CITATIONS
SEE PROFILE

Mirko Suznjevic
University of Zagreb
**56** PUBLICATIONS **632** CITATIONS
SEE PROFILE

Eusebi Calle
Universitat de Girona
**92** PUBLICATIONS **944** CITATIONS
SEE PROFILE

**Multimedia Systems manuscript No.**
(will be inserted by the editor)

# Understanding and Modelling the Network Traffic of the OnLive Cloud Gaming Platform

**M. Manzano · M. Urueña · M. Sužnjević**
**E. Calle · J. A. Hernández · M. Matijasevic**

Received: date / Accepted: date

**Abstract** *Cloud gaming* is a new paradigm, which is to play a pivotal role in the video game industry in the forthcoming years. Cloud gaming, also called gaming on demand, is a type of online gaming that allows on-demand streaming of games onto a non-specialised hardware (i.e., PC, smart TV, etc.). This approach does not require downloads because the actual game is stored on the game company's server and is streamed directly to the client. Nonetheless, this revolutionary approach significantly affects the network load generated by online games. Cloud gaming presents new challenges for both network engineers and research community, who should gain knowledge regarding these new cloud gaming platforms. The purpose of this paper is to shed light into OnLive, one of the most popular cloud gaming platforms. Our major contributions are: (a) a review of the state of the art of cloud gaming; (b) reverse engineering of the OnLive protocol; and (c) a synthetic traffic model for OnLive. To do so, extensive traffic traces have been collected and studied. Currently, there is no detailed study available describing the protocol of any cloud gaming platform. Therefore, we believe that this study will lay a foundation for the future cloud gaming research.

**Keywords** Cloud gaming · Online games · OnLive · Reverse Engineering · Traffic modelling

M. Manzano and E. Calle
Institute of Informatics and Applications (IIiA), University of Girona, Spain
E-mail: {mmanzano,eusebi}@eia.udg.edu

M. Urueña and J. A. Hernández
Dept. of Telematic Engineering, Universidad Carlos III de Madrid, Spain
E-mail: {muruenya,jahguti}@it.uc3m.es

M. Sužnjević and M. Matijasevic
Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia
E-mail: {mirko.suznjevic,maja.matijasevic}@fer.hr

## 1 Introduction

Although nowadays the traffic generated by networked gaming (i.e., by video game platforms such as PC, Xbox 360, PlayStation 3 or Nintendo Wii) only represents 0.03% (67 PB) of the total Internet traffic, it is expected to experience a 46% growth in 2016 [8], becoming the Internet's traffic segment with the highest expansion in the upcoming years. Online gaming brings together players from all over the world together for fun and entertainment, and has been regarded as one of the most profitable and popular Internet services. As a consequence, the business of video games is expected to grow from $60.4 billion in 2009 to $70.1 billion in 2015 [13].

In a broader context, there is a growing trend towards moving local applications to remote data centers: this is often referred to as the *cloud*. This new computing paradigm provides several well-known benefits such as: (a) a decrease in terms of hardware and software requirements for clients (enabling the so-called *thin clients*). Users no longer need to install programs locally and carry out periodic updates, administration or maintenance tasks for their computers; (b) users may access their files, e-mail, music or movies from any device (e.g. PC, smartphone, tablet, smart TV, etc.) at any time and anywhere; and (c) developers do not need to devote resources to adapt their software to different hardware platforms.

Many types of services focusing on this new paradigm have appeared in the last years: cloud storage (e.g. Dropbox, Google Drive), music-on-demand (e.g. Spotify, Pandora), video-on-demand (e.g. Youzee, Netflix), document edition and spreadsheets (e.g. Google Docs) and more recently, cloud gaming (e.g. OnLive, Gaikai), among others.

From the network perspective, the cloud paradigm has a direct impact on the network load [43]. In the area of networked games, the traffic requirement of cloud games in comparison with standard games may increase by a factor of 100. To illustrate this point, World of Warcraft, a popular online game, has a bandwidth requirement around 55 Kbps in the most demanding situations [36], while OnLive can require up to 5.6 Mbps [26]. While cloud computing reduces the hardware requirements of clients, it may significantly increase the network requirements for a good Quality of Experience (QoE, a subjective measure of the end-users with respect to a given service) of such services, since many of them are traffic-intensive. In fact, some business models like Spotify greatly rely on the performance of the network [16]: *premium* users require higher network capabilities than *free* users, since they receive higher quality streams.

In this paper we focus on online cloud gaming (or *Gaming on Demand*). At the time of writing, there are only a few cloud gaming service platforms such as OnLive, Gaikai, GameAnywhere, GameString, G-cluster, Otoy, StreamMyGame, t5 labs or iTSMY, and some of them are still under development. Recently, Gaikai has been purchased by Sony for $380 million [39]. The fact that some of the strongest video game companies are now investing on cloud gaming suggests that this new technology is to become a considerable share of the total amount of the video game industry.

In this light, network operators must design their infrastructures in order to provide access technologies capable of dealing with the network requirements of cloud gaming (as well as other cloud-based services), since current access technologies (i.e., 802.11, xDSL, HFC) may no longer be suitable to deliver high-quality cloud services.

In our previous work [26], we carried out a traffic measurement comparison between OnLive and Gaikai and concluded that there are several significant differences between these platforms. However, to the best of our knowledge, there are no previous works focusing on the specific protocols used by such cloud gaming platforms. Consequently, we have focused our work to conduct a reverse engineering study on OnLive, leaving Gaikai to be studied in the future.

This paper has two main objectives: (1) to study the OnLive protocol, based on extensive traffic traces of five games; and (2) to propose a OnLive traffic model which can be used for network dimensioning, planning optimization and other studies. The importance of these models is further increased due to the fact that cloud gaming traffic can be currently classified as one of the most demanding types of network traffic because of very high bandwidth loads as well as very low latency requirements.

The rest of this paper is organised as follows. Section 2 presents previous works in the literature related with cloud gaming and traffic modelling for online games. Section 3 thoroughly describes the OnLive protocol. Then, the measurement methodology and the characteristics of the main traffic flows of the OnLive protocol are discussed in Section 4. Section 5 presents the traffic generation model and provides an experimental validation. Finally, conclusions and further work are discussed in Section 6.

## 2 Related works

Since the inception of Internet, its traffic has evolved and nearly doubled every year [27]. In the 90s, web and e-mail traffic accounted for the majority of exchanged traffic. In the past decade, peer-to-peer and streaming traffic took the lead with the advent of BitTorrent and YouTube. Nowadays, social networks and cloud computing are becoming very popular and are continuously increasing their portion of the total traffic. For instance, in the first half of 2013, Netflix accounted for almost a third of the total downstream traffic in North America [34].

Regarding video games, several studies have previously analysed the impact of online gaming on the network. In [15] the authors studied several popular online games. For instance, it was shown that Unreal Tournament 2003 could be played by using 28.8 Kbps modems. Furthermore, the authors of [4] studied the traffic load produced by the widely popular Call of Duty: Modern Warfare 2 game and reported that server-originated traffic produces only 34 Kbps, while the client-originated one is about 13 Kbps. Later studies analysed the well-known video game World of Warcraft and reported that it only requires a 5 Kbps uplink and a 55 Kbps downlink [36], thus coming out to the same conclusion: traditional online video gaming is not a bandwidth-hungry application.

Cloud computing has recently emerged as a new paradigm for hosting and delivering services over the Internet, and has appeared to be a remarkable opportunity for the video game industry. The authors of [20] have defined four requirements for cloud-based gaming services:

- *User responsiveness*: the interaction in the games should be kept below 80 ms in order to guarantee suitable responsiveness [9].
- *High-quality video*: in order to provide high quality video, interactive video (1280 x 720 resolution), data shall be reduced as much as possible while keeping quality.
- *Quality of Service (QoS)*: in order to decrease the latency and avoid network bottlenecks, game traffic should be prioritized. An example of such a technique was presented in [3].
- *Operating costs*: it is necessary to develop optimization techniques to minimize power consumption.

In this same paper, the authors defined two approaches for game streaming:

- *Graphics streaming*: directly transmits the graphic commands to the client device and the rendering of the image is carried out by the client device [17].
- *Video streaming*: the server renders the game graphics scene, the framebuffer is captured, eventually downsampled to match the target resolution and encoded using standard video codecs such as MPEG-2, MPEG-4 or H.264 [6], [18]. This approach is the most suitable for thin clients because they lack hardware accelerated rendering capabilities [41]. Therefore, this approach is the most used one by cloud gaming platforms.

In [32] the authors evaluated the quality of streamed online gaming over fixed WiMAX and concluded that the quality of streamed gaming is very sensitive to delays and that the transmission latencies of WiMAX are near the edge of a smooth gaming experience. In fact, it was noted that under heavy traffic loads, the gaming experience faced a radical degradation.

The authors of [5] performed a delay comparison analysis of OnLive and StreamMyGame, and identified that OnLive implements a game-genre-based differential resource provisioning strategy to provide sufficiently short latency for real-time gaming. Nevertheless, the work did not go further into the characterisation of basic traffic metrics of online games.

Unlike YouTube, OnLive is a real-time stream, which means that the video source is created as it is streamed. Therefore, no traditional compression techniques can be applied to the entire video. As a consequence, it must compress the video in real-time with the purpose of reducing the bandwidth requirements (i.e., with the standard H.246), albeit increasing the latency.

Furthermore, a QoE analysis of three game genres was carried out in [25]: FPS (first-person shooter), RPG (role-playing games) and ACT (action). The paper concluded that the most sensitive games with respect to *real-time strictness* (a metric defined in the paper) were FPS, followed by RPG and ACT, in second and third places, respectively.

The work presented in [10] analysed the traffic characteristics of the OnLive cloud gaming platform, under different network parameter scenarios (capacity, latency and packet loss), and compared it with the traffic obtained from YouTube and a Skype video call. It concluded that OnLive is a fundamentally different service than Skype and YouTube (in terms of network performance metrics) and that OnLive adapts to possible changes on network conditions (i.e., delay or bandwidth).

Finally, as previously mentioned, in [26] the authors analysed the traffic characteristics of OnLive and Gaikai. It was pointed out that both platforms were similar regarding their packet size distribution, but significantly different regarding their packet inter-arrival times. Furthermore, it was also observed that traditional online gaming is remarkably different from cloud gaming in terms of network load and traffic characteristics.

The next step following the traffic analysis of video games is creating models of network traffic, that can be used for a variety of purposes related to designing network infrastructures, architectures, protocols, etc. Research efforts in the area of traffic modelling for online games have closely followed the development of the online game market and the popularity of certain game genres.

Initially, most works in this area have been focused on the first-person shooter (FPS) genre and follow the modelling approach pioneered by Paxson [30]. The first work of traffic modelling for games was done by Borella [2], who studied the traffic of a popular FPS: *Quake I* and its sequel *Quake II*. The author modelled the empirical game traffic with analytical models of packet inter-arrival times (IAT) and packet size (PS), while also taking into account the hardware capabilities of the computers on which the game was run. The following works were focused on different games.

Lang et al. measured, analysed, and modelled the network traffic of several FPS games. In [24] the authors modelled the traffic of *Quake III*. In addition to modelling PS and IAT, the impact of different graphic cards, number of the players, and maps (i.e., virtual worlds on which the game takes place) on PS and IAT were investigated. The traffic of the game *Halo* by Microsoft, for the Xbox console was also analyzed and modelled [22] as well as the traffic of *Half-Life* [23] in which they examined OpenGL versus software-based rendering and its impact on network traffic. Färber [14] analysed traces of *Counter Strike*, which were gathered during a 36 hour LAN party involving 37 players, and created a traffic model dependent of the number of active clients. A different kind of modelling approach was presented by Cricenti and Branch [11] who modelled the traffic of *Quake 4* using mixed autoregressive/moving average (ARMA) models.

Not only FPS games have been studied, but also Real Time Strategy (RTS), such as *Starcraft* [12], as well as Massively Multiplayer Online Role-Playing Games (MMORPGs) later on, after the increase of their popularity among the player base. Svoboda et al. [38] analysed traffic traces captured within the 3G mobile core network and noted that WoW was among top 10 TCP services in the monitored network and consumed 1% of all TCP traffic. They also provided a traffic model for WoW. Wi et al. [42] modelled the traffic of one of the most popular MMORPGs in China - *World of Legend*, while Kim et al. [19] presented traffic models of *Lineage*.

As the traffic of MMORPGs is highly variable and difficult to model, new approaches for traffic modelling have appeared in the literature. The authors of [35] developed a novel transformational modelling procedures and compared them to standard modeling techniques. A different approach taken by several research groups was the classification of user behaviour at application level and the modelling of separate behaviours. All classification approaches [40,29,37] used WoW as a case study.

Cloud gaming is still a relatively new technology; thus, there are still several aspects that need to be investigated in detail. To the best of our knowledge, no reverse engineering of the protocols used by cloud gaming platforms and no traffic models have been yet proposed. Therefore, in this paper we carry out a reverse engineering of the OnLive protocol and we propose a network traffic model for this platform. The main purpose is to provide the research community and network engineers the knowledge about the network load caused by these new emerging applications, which will help to design future network infrastructures. For instance, network providers could prioritize specific flows in order to enhance the QoE of end users. In addition, the research community could also use these models all kind of tests regarding new protocols, applications, network infrastructures, QoE studies, etc.

## 3 The OnLive Protocol

*OnLive* is a cloud gaming platform available for Windows and Mac OS X. Additionally, it is also supported by Android and iOS devices (smartphones and tablets) and some recent smart TVs, from vendors like LG and Sony. According to the OnLive's website [28], for a good QoE the platform requires a minimum bandwidth connection of 2 Mbps to render content at 1024 x 576, recommends 5 Mbps for 1280 x 720, and is able to output 1080p with a higher bandwidth connection. OnLive requires a local installation of its own software client.

We have studied the behaviour of the OnLive protocol by analysing the packet traces perviously employed to characterize the aggregated OnLive traffic [26]. A more careful inspection has let us identify the different flows that compose such traffic, and the protocols employed to transport them. To identify the purpose of each flow we have carried out additional experiments in the same scenario described in Section 4.1, where we have selectively filtered individual flows to observe its effect on the gaming session.

It must be noted that, although we use the "OnLive protocol" term, the OnLive platform employs a number of different control and data protocols, transported using Transport Layer Security (TLS) over TCP connections and RTP/UDP flows, which are employed for a variety of purposes during the different phases of an OnLive session.

We have identified three main phases in an OnLive session. In the first one the *OnLive Client* authenticates (using a TLS/TCP connection) and measures the latency and available bandwidth with multiple OnLive Servers. In the second phase, once a suitable server is selected by the client, it starts streaming the OnLive menu. Finally in the third phase, the client selects a video game and starts playing. The OnLive menu and the playing session are streamed in a similar way, utilizing multiple RTP/UDP flows multiplexed over a single UDP port (the default OnLive UDP port is 16384).

We will now describe the most relevant details of each phase comprising an OnLive session.

### 3.1 Phase 1: Authentication and Global load balancing

When the OnLive client software is executed, it connects to one of the *OnLive Authentication Servers*[1] listed under the DNS name `ds.onlive.net` by using a TLS/TCP encrypted session (port 443). Therefore we are unsure of the messages exchanged through such connection, although they are probably HTTP-based and involve authenticating the user, as well as probably obtaining the IP addresses of several OnLive servers that are employed in the next latency measurements.

At the time this paper is being written (March 2013), OnLive has servers in at least 4 different locations (i.e.,

[1] To simplify the description of the OnLive session, we will give a name to the different protocols and roles of OnLive Servers that have been identified, although the different roles may be also performed by the same physical server.

London, Washington, Dallas and San Jose in California), which allows OnLive to cover most USA and western Europe within a 1000 mile radius, which OnLive considers close enough to have good latency, although other studies claim otherwise [7]. Once authenticated, the client starts measuring its latency with those locations in order to find the most suitable OnLive server, given the importance of having a low latency with the server for game responsiveness.

These latency measurements are performed using what it seems a custom probing protocol conveyed in short RTP/UDP messages. In particular the client starts sending batches of 9 RTP messages every 22 ms to 9 different IP addresses (UDP port 16384) belonging to 2 or 3 *OnLive Probing Servers* in each of those 4 locations. Each RTP message comes from a different UDP port of the client and has a different Synchronized Source Identifier (SSRC) value, but all share the 0x2800XXXX prefix, which as we will see later is employed by the *OnLive Probing Protocol*. The flow of RTP messages with each particular Probing Server use consecutive Sequence Numbers and a Timestamp with the number of microseconds since the client measurement session started. Although the Payload Type is set to 0, which is reserved for G.711 PCMU voice samples, instead the RTP payload carries custom protocol messages where the first 32 bits seem to identify the particular type of message[2].

The OnLive Probing Servers reply back to the client with similar RTP messages, changing the payload but keeping the same SSRC, Sequence Number and Timestamp than the client messages they are echoing. With this information the client may easily measure the Round-Trip Time (RTT) with each server, as well as to detect lost packets. After a fixed number (i.e., 12) of RTP exchange rounds since the first response from each server, the servers send a final RTP message with a slightly different SSRC=0x28YYXXXX that signals the end of the measurement session. Since we do not have access to the source code of the OnLive client software, we do not know the exact details of the server selection algorithm but is seems to use the RTT as its main metric, since the client systematically selects one of the first responding servers, which are also the ones finishing the measurement session first.

At this moment, after estimating the RTT with each server, the OnLive Client chooses the “closest” one and starts a TLS/TCP session (port 443 again) with it. Then, after exchanging a couple of encrypted messages, the client starts another RTP-based measurement session with that OnLive Probing Server. This measurement session starts in a similar way to the previous one, since it also employs a SSRC=0x2800XXXX identifier (although different to the previous one) and the short RTP messages from the client are echoed by the server (16384 port), but now the RTP messages are sent every 4 milliseconds and last 200 rounds. Moreover, when this latency measurement session is over, the client sends a RTP message to a different UDP port[3] of the server to start a RTP flow to measure the client’s downstream bandwidth.

Although this RTP flow keeps the same SSRC as the previous one, now these RTP messages are only sent by the OnLive Probing Server and are not replied back by the client. These packets are quite large (1400 bytes) and sent every 1.3 milliseconds until the 2292 sequence number is reached (i.e., 3 seconds), when the server stops sending. There is a final exchange of control RTP messages where the server notifies to the client its public IP address and port, probably to identify intermediate NATs. It is worth noting that all OnLive RTP flows and TCP connections are initiated by the client in order to avoid NAT and firewall issues.

After these two measurement sessions with the selected OnLive Probing Server are over, the client may now estimate both the round-trip time and the available downstream bandwidth. If the client detects that the measured performance is below the minimum requirements, it shows a warning message to the user, and may adapt the streaming bit rate as reported by Claypool et al. [10].

The client probably notifies the values of these measurements back to the OnLive Probing Server by means of the TLS/TCP connection. This message from the client triggers some internal operations with other OnLive Servers, because the response takes several seconds to be returned by the Probing Server. When the TLS/TCP response finally arrives, the client starts the following phase of the OnLive session to show the main menu.

## 3.2 Phase 2: OnLive Main Menu

The main menu of OnLive is not a web page or not even a Flash Player plugin but an interactive video that employs the same streaming protocol that is later used to actually play the games. We will refer to this protocol as the *OnLive Streaming Protocol*.

This phase starts as soon as the OnLine Probing Server employed in the previous phase replies back to

[2] We have identified more fields and the different messages types of this custom protocol, but they are not described here for sake of clarity.

[3] This port is dynamic and greater than the OnLive’s 16384 default one. It is notified to the client using the custom protocol encapsulated in echo RTP messages from the server.

the TLS/TCP request of the OnLive Client. The response should include the IP address of the OnLive Menu Server, which is collocated (i.e., has the same IP \24 prefix) with the selected Probing Server but is probably a different machine (i.e., has a different IP address). At that moment the client sends a RTP message to the new *OnLive Menu Server*.

The RTP messages of this phase, either from the client or from the servers, are different from the ones employed by the OnLive Probing Protocol (i.e., SSRC= 0x28XXXXXXXX). In particular, in this phase there are multiple RTP flows multiplexed in the same UDP flow, both in downstream (server-to-client) and upstream (client-to-server) directions. Then the different RTP flows can only be identified by their SSRC values. Table 1 shows all the RTP flows, which for the sake of clarity have been named as indicated by the *Acronym* column. From now on, this document addresses the different flows by their acronym. As observed, downstream flows from the server employ fixed SSRC values while clients generate different SSRCs for each OnLive session, although they have a similar format (SSRC=0x000YXXXX) where only the 2 last bytes change (i.e., 0xXXXX), while the first two bytes (i.e., 0x000Y) identify the type of upstream flow. Most flows have a dynamic payload type of 100, but the video and one of the audio flows have a 96 and 101 payload type respectively. Each RTP flow keeps its own Sequence Number and Timestamp values, although in most of them the Timestamp field is set to zero. However, the RTP messages of the OnLive Streaming Protocol also include a Header Extension with three 32 bits-long fields that contain a microsecond timestamp, and a common counter of all RTP messages generated by the source client/server, irrespectively of the RTP flow they belong to. This timestamp field in the RTP extension header has allowed us to precisely measure the inter-arrival time of OnLive packets without the jitter generated by the network.

Although, as already mentioned, both the OnLive menu and the gaming session employ exactly the same Streaming Protocol, including all flows listed in Table 1, in this subsection we will only explain the flows related to control functions, while the game-related flows will be explained in the next subsection.

When the OnLive Menu Server receives the first RTP message from the client, it starts the Monitor flow that is employed to continuously track the session's Quality of Service. Whenever a OnLive menu or gaming session begins, this flow starts with a burst of 4 long RTP messages that is followed by a small (i.e., 48 bytes) RTP message sent 10 milliseconds later. This sequence is repeated several times, probably to measure the available bandwidth with the new OnLive Menu Server (if it does not reach the minimum of 2Mbps as stated in the official website, the OnLive client asks to reconnect). Then, until the end of the phase, the RTP flow only sends one small RTP packet every second, probably to continuously measure the latency to the server (the client gives a warning when these packets are filtered).

The next RTP flows that appear are the Control and Control-ACK ones, sent by the server and the client respectively. Actually these two flows work together in what we have interpreted as some kind of control protocol. When the server wants to convey some message to the client it sends a RTP packet with SSRC=0x10000, that the client acknowledges with a fixed-size (i.e 86 bytes-long) RTP packet with SSRC=
0x4XXXX. For reliability the server keeps sending the same payload every 8.25 ms until it receives a response from the client (which acknowledges each message). These control message exchanges occur sporadically, for instance at the beginning and ending of the menu or gaming phases, or when the user performs some selection in the OnLive main or in-game menus that should be notified back to the client, such as configuring the audio or muting[4] it.

The client notifies the actions of the user to the servers by means of the Keys flow, which will be described later since is mostly employed in the gaming phase, as well as with the Mouse and Cursor flows that are related to the mouse pointer. Instead of the client sending a message to the OnLive server whenever the mouse is moved, these two flows have a fixed rate of one RTP message every 50 milliseconds, and the packets of both flows are interleaved (i.e., there is a 25 ms gap between a client packet and a server one and vice versa). The size of the server's Cursor flow payload is fixed (i.e., 16 bytes) whereas the client's Mouse messages have a variable size. Therefore the client may send multiple mouse movements per message, while the server acknowledges them and may reply with the current cursor position.

This phase ends when the user selects a game to play in the OnLive menu. At that moment the OnLive client stops communicating with the OnLive Menu Server and opens a new UDP socket to send a RTP message to the *OnLive Gaming Server* that also uses the OnLive default 16384 port. In most of the captured sessions the Menu and the Gaming Servers have consecutive IP addresses, and in some cases they have the same IP address. Thus we wonder whether OnLive Menu Servers and OnLive Gaming Servers work in pairs, or they are

[4] Quite interestingly the muting audio option does not interrupt the audio flows, but just notifies the client to do not play them back.

**Table 1** RTP/UDP flows of the OnLive Streaming Protocol

| Direction | RTP SSRC | RTP Payload Type | Acronym | Flow description |
|---|---|---|---|---|
| Downstream | 0x00000000 | 100 | Monitor | QoS monitoring flow |
| Downstream | 0x00010000 | 100 | Control | OnLive Control |
| Downstream | 0x00030000 | 100 | CBR-Audio | Audio stream (CBR Codec) |
| Downstream | 0x00040000 | 100 | Cursor | Cursor position |
| Downstream | 0x00050000 | 101 | VBR-Audio | Audio stream (VBR Codec) |
| Downstream | 0x00060000 | 96 | Video | Video stream |
| Downstream | 0x00080000 | 100 | Chat | Voice Chat (Sound from other players) |
| Upstream | 0x0000XXXX | 100 | Keys | User input (keyboard and mouse buttons) |
| Upstream | 0x0001XXXX | 100 | Mouse | Cursor movement |
| Upstream | 0x0004XXXX | 100 | Control-ACK | OnLive Control ACK |
| Upstream | 0x0008XXXX | 100 | Mic | Voice Chat (Microphone from the user) |

actually the same server, although we have not been able to verify any of these hypotheses.

## 3.3 Phase 3: Playing a game

The gaming phase also employs the OnLive Streaming Protocol, including the QoS monitoring, control and mouse pointer flows described in the previous subsection. We now complete the description of this protocol by explaining the remaining RTP flows of Table 1, which are the most important flows of this phase and of the whole OnLive Streaming Protocol.

In particular the Video flow has the highest rate because it carries the OnLive video stream. The RTP messages of this flow have a dynamic Payload Type (i.e., PT=96), with consecutive Sequence Numbers. However the Timestamp field also carries a monotonically increasing counter that has the same value for consecutive RTP messages and thus may identify a burst of packets. Moreover the last packets of a burst are signalled by setting the RTP Mark flag. The rate of the flow varies depending on the movement and complexity of the frames, thus it surely employs a variable bit rate (VBR) video codec. Some sources[5] suggest that OnLive employs a modified H.264 codec, but we have not been able to verify this.

The audio stream is not integrated in the video RTP flow but in a separate one. Actually we have identified two audio RTP flows (CBR-Audio and VBR-Audio) from the server. The purpose of these two flows is not completely clear to us because they are streamed in parallel, but they seem to carry the same audio stream (i.e., the audio keeps playing when dropping any of them, unless both are filtered simultaneously). Furthermore, each flow seems to use a different codec and even RTP encapsulation. The CBR-Audio messages have a 100 Payload Type value, the Timestamp field is set to zero, and carry a fixed 204 bytes payload that is sent every 10 milliseconds. The VBR-Audio flow has a 101 Payload Type, a Timestamp equal to the Sequence number and carries longer, variable payloads sent every 50 ms. Maybe this flow is related to the 5.1 surround audio supported by some games, but we haven't been able to verify this because we have not been able to identify the audio codecs.

The user actions, other that moving the mouse cursor explained in the previous subsection, are conveyed to the server in the Keys flow, with payloads ranging from 28 to 60 bytes, which are sent on-demand when the user presses a key or button.

The last two RTP flows of the OnLive Streaming Protocol are related to the Voice Chat functionality that allows OnLive users to talk among them. Then the Mic flow carries the user's microphone stream to the OnLive server, while the Chat flow from the server carries the audio from other users to the local client.

When the gaming session is over, and the user returns to the OnLive Menu (phase 2), the client again stops communication with the previous server (e.g. OnLive Gaming Server) and employs a new UDP port to communicate with the new one (e.g. OnLive Menu Server).

Since this gaming phase is the most important one of an OnLive session, the remaining of the paper will focus on it, analysing in detail the traffic characteristics of each flow as well as the aggregated traffic, in order to create a synthetic OnLive traffic model.

## 4 Traffic Characteristics of OnLive

This section presents the measurement methodology followed in order to carry out the reverse engineering of the OnLive protocol and the traffic characteristics, which are divided in three subsections. The first subsection explains the measurement scenario and methodology. Subsection 4.2 gives an overview of the results obtained from the measurements carried out. Then, sub-

[5] http://forum.doom9.org/showthread.php?t=151730

sections 4.3 and 4.4 are focused on the last phase of the OnLive protocol (i.e., playing a game). In both steps we present the server- and client-originated traffic respectively for two specific games, which have been selected because they have shown the two most different traffic profiles [26].

**Table 2** OnLive games considered in this paper

| Game | Nickname | Genre |
|---|---|---|
| Pro Evolution Soccer 2012 | *pes2012* | Sports |
| Unreal Tournament 3 | *unreal3* | FPS |
| Crazy Taxi | *crazytaxi* | Racing |
| Air Conflicts | *airconflicts* | Flight simulator |
| 4 Elements | *4elements* | Puzzle |

### 4.1 Measurements

We have captured traces for five different games of the OnLive platform in order to identify the main features of its traffic pattern. Table 2 summarises the games selected with their nickname and genre.

Traffic traces have been captured using Wireshark[6] at the local computer of the gamer. Measurements have been carried out from Spain by using a wired University connection (100 Mbps Fast Ethernet) access, with 94.17 Mbps of downstream, 71.81 Mbps of upstream, 1 ms of jitter and 38.12 ms of RTT (Round-Trip Time) to the nearest OnLive data center. The access network characteristics have been obtained using *pingtest*[7] (jitter) and *speedtest*[8] (downstream and upstream bandwidth), plus the *traceroute* command-line tool (RTT). In [26], the same traffic characteristics were also obtained from a common wireless home connection (802.11n + VDSL), however in this work we have only considered the former network scenario.

The capture of traffic traces has been started, for each game, at the time of executing the client program and finalised after playing during 100 seconds. Because the time required to load each game is different, the traces' total times differ in each case.

From each traffic trace we then obtain: (a) the packet size; and (b) the inter-arrival time of all captured packets.

Thereafter, we have thoroughly analysed the traffic traces in order to carry out the reverse engineering of the OnLive protocol, as well as live experiments where specific RTP flows were filtered by their SSRC field using the u32 binary match filter of iptables.

[6] http://www.wireshark.org/
[7] http://www.pingtest.net/
[8] http://www.speedtest.net/

### 4.2 Overview

Here, a detailed summary of the overall measurement results is presented in Table 3. The table shows for each video game the measured traffic metrics. Traffic metrics are separated in *downstream* and *upstream* directions. It is very interesting to note the notorious difference between the bandwidth required by this cloud gaming platform and previous online games such as Call of Duty (34 Kbps in downstream and 13 Kbps in upstream) [4]; World of Warcraft (which only goes up to 55 Kbps in downstream and 5 Kbps in upstream) [36] and Unreal Tournament 2003 (could be played by using 28.8 Kbps modems) [15,33].

In next subsections, the server- and client-originated traffic of *crazytaxi* and *4elements* is provided because they show the most different bandwidth values (see Table 3). In both cases we have considered the last phase of the OnLive protocol, because it is the most significant one in terms of traffic load.

Table 4 presents a dissection of the different RTP flows which have been observed in the last phase of the OnLive protocol. As previously showed in Table 1, it can be observed that there are seven different RTP flows in the downstream, while there are four in the upstream. It is worth highlighting the fact that the CBR-Audio and Cursor flows have constant packet size and packet inter-arrival times. Although VBR-Audio is the flow composed of the largest packets, it can be observed that the Video flow is the flow which comprises the majority of the bandwidth used by OnLive.

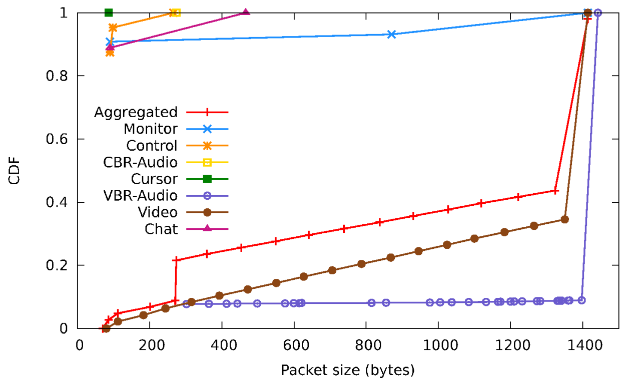
### 4.3 Downstream OnLive Traffic (Server $\rightarrow$ Client)

The measurements regarding the server-originated traffic (*downstream*) are analysed next. For both games the cumulative distribution functions (CDF) of packet sizes and packet inter-arrival times are shown in Figures 1 and 2. Results of the aggregated traffic of the last phase of the OnLive protocol are compared with all the particular RTP flows generated by OnLive in that phase.

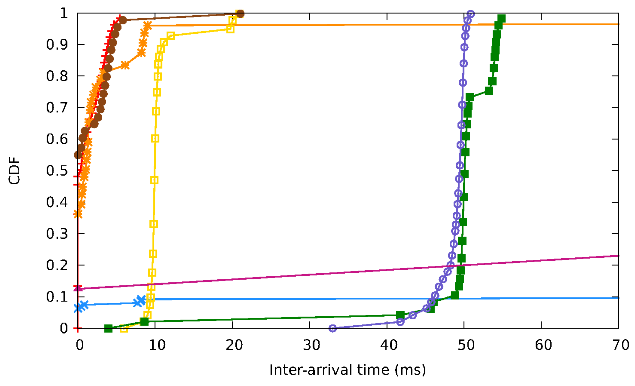
Fig. 1a depicts the CDF of packet size (in bytes) of *crazytaxi*. As observed, the aggregated curve is presented in order to compare it with the dissected RTP flows. The Monitor flow line indicates that almost all packets of this flow are very small. This effect can also be observed in the case of the Control flow. Then, it is worth noting that both CBR-Audio and Cursor flows exhibit a constant packet size: the former around 270 bytes and the latter around 90 bytes. The VBR-Audio flow is mainly composed of packets of Ethernet-like MTU size (i.e., 1418). Furthermore, it can be observed

**Table 3** Summary of the traffic characteristics of five OnLive games.

| | | *pes2012* | *unreal3* | *crazytaxi* | *aircombat* | *4elements* |
|---|---|---|---|---|---|---|
| Downstream | Total time (s) | 249.41 | 261.16 | 200.56 | 239.68 | 236.59 |
| | Number of packets | 149004 | 174265 | 13867 | 153798 | 108619 |
| | Avg. packets / sec | 597.41 | 667.25 | 691.39 | 641.66 | 459.09 |
| | Avg. packet size (B) | 915.57 | 975.05 | 1014.99 | 955.65 | 722.58 |
| | Bit rate (Mbps) | 4.37 | 5.21 | 5.61 | 4.91 | 2.65 |
| Upstream | Total time (s) | 249.48 | 261.31 | 200.69 | 239.83 | 236.72 |
| | Number of packets | 8947 | 13943 | 6825 | 14677 | 14849 |
| | Avg. packets / sec | 35.86 | 53.35 | 34.01 | 61.19 | 62.72 |
| | Avg. packet size (B) | 168.49 | 157.8 | 170.08 | 154.81 | 154.91 |
| | Bit rate (Mbps) | 0.0048 | 0.067 | 0.046 | 0.075 | 0.077 |



(a) CDF of packet size (bytes)



(b) CDF of packet inter-arrival times (ms)

**Fig. 1** Downstream traffic characteristics of *crazytaxy*

that the Video flow holds a pivotal role regarding the CDF of the aggregated traffic. In that flow, while 65% of packets are of MTU size, the other 35% present a uniform distribution from 50 bytes to the MTU. Thus, it can be inferred that the Video flow is composed of bursts which can not fit in a common MTU-sized packet, forcing to split[9] each of the bursts in several packets: the first ones of size equal to MTU and a last one of a smaller size. The Chat flow is mostly composed of small packets.

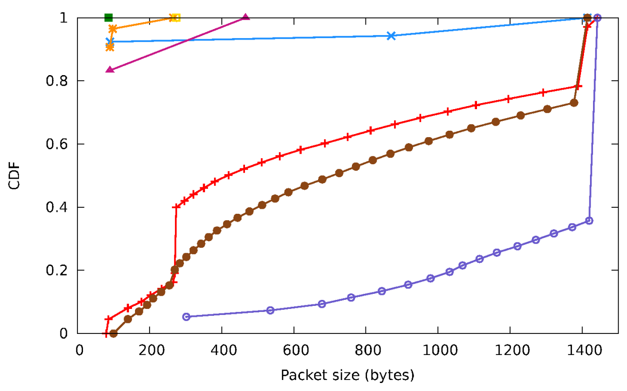
**Table 4** Summary of the characteristics of the RTP flows observed in the last phase of the OnLive protocol. Values are displayed as following: *crazytaxi* / *4elements*.

| | | Avg. packet size (Bytes) | Bit rate (Kbps) |
|---|---|---|---|
| Downstream | Aggregated | 1017.61 / 688.38 | 6020 / 2450 |
| | Monitor | 198.62 / 180.51 | 2 / 1 |
| | Control | 98.88 / 96.6 | 1 / 14 |
| | CBR-Audio | 274 / 274 | 200 / 200 |
| | Cursor | 86 / 86 | 10 / 10 |
| | VBR-Audio | 1348.15 / 1250.25 | 220 / 210 |
| | Video | 1157.43 / 786.69 | 5632 / 2020 |
| | Chat | 131.78 / 152.67 | 0.1 / 1 |
| Upstream | Aggregated | 175.10 / 154.89 | 70 / 120 |
| | Keys | 175.11 / 155.09 | 50 / 80 |
| | Mouse | 221.65 / 233.24 | 20 / 40 |
| | Control-ACK | 86.65 / 86.80 | 0.1 / 3 |
| | Mic | 253.64 / 315 | 0.2 / 0.4 |

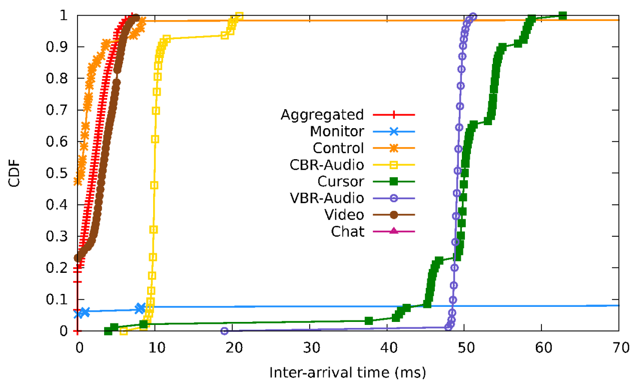
Regarding the inter-arrival times (in milliseconds) of *crazytaxi*, Fig. 1b shows several important aspects, which help to understand the underlying structure of the OnLive Protocol. In this case the CDF of the aggregated traffic is also provided. As can be observed, Monitor and Chat are the two flows which send packets with the least frequency; more than 70 ms (not shown in Fig. 1b for the sake of simplicity). On the contrary, both CBR and VBR flows and the Cursor flow are predominantly constant in their inter-arrival times; being them around 10 ms, 50 ms and 10 ms, respectively. Then, it can also be observed that the OnLive Control flow packets are mostly sent within the 0 - 10 ms period. Finally, it is important to note that the packets of the Video flow are the most frequent ones, being the majority of them sent within a 0 - 6 ms period.

The CDF of packet size of *4elements* is shown in Fig. 2a. It is worth mentioning that the most RTP flows (Monitor, Control, CBR-Audio, Cursor and Chat) present very similar curves to the ones obtained for *crazytaxi*. However, as can be observed, VBR-Audio and Video are significantly different from the same RTP flows of *crazytaxi*; consequently varying the CDF of

[9] This splitting is performed at the application layer, that is, several RTP messages are generated instead of being fragmented at the IP layer.

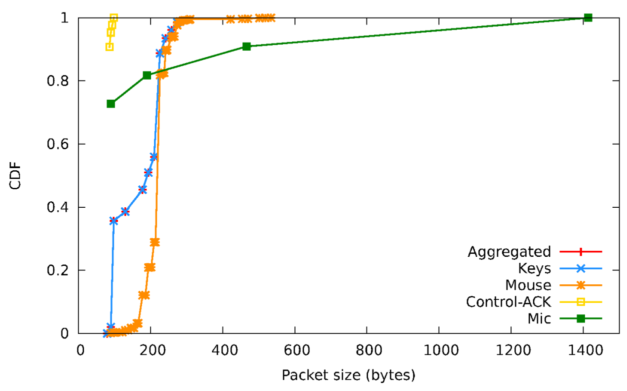


(a) CDF of packet size (bytes)

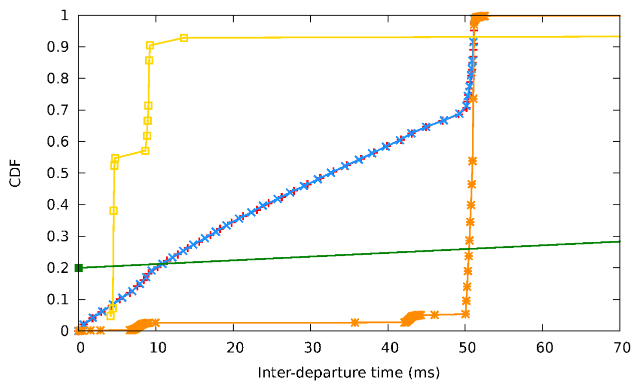


(b) CDF of packet inter-arrival times (ms)

**Fig. 2** Downstream traffic characteristics of *4elements*



(a) CDF of packet size (bytes)



(b) CDF of packet inter-departure times (ms)

**Fig. 3** Upstream traffic characteristics of *crazytaxy*

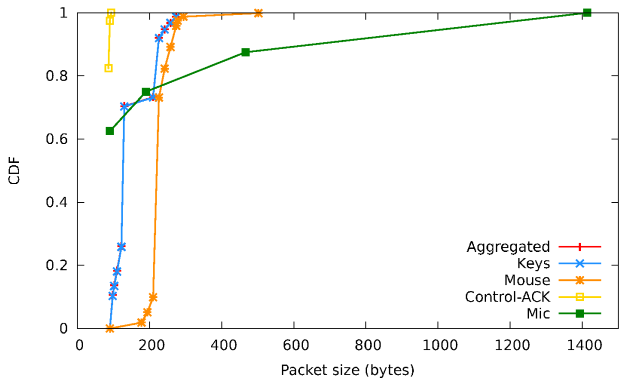
the aggregated traffic. While the 90% of packets of the VBR-Audio flow of *crazytaxi* are of MTU size, only around 60% of packets show this MTU size in the case of *4elements*. Moreover, the portion of MTU size packets from the Video flow of *4elements* is about the half than the ones of *crazytaxi*. As a consequence, it can be inferred that the traffic generated by the VBR-Audio and the Video flows are dependent on the game that is being played.

Finally, Fig. 2b presents the CDF of inter-arrival times (in ms) of *4elements*. As observed in the case of *crazytaxi*, the Monitor and Chat flows of *4elements* are the least frequent ones (the latter does not appear in Fig. 2b because it has a high value of inter-arrival time). In addition, CBR- and VBR-Audio flows and the Cursor flow are subjected to specific inter-arrival times, being them 10 ms, 50 ms and 10 ms, respectively. It is interesting to note that for *4elements*, the VBR-Audio flow shows more inter-arrival time modes than for *crazytaxi*. The Control flow is roughly the same than the observed in *crazytaxi*. To conclude, it is shown that the Video flow is composed of a higher number of small packets and thus shorter bursts than the same flow of *crazytaxi*. This difference might be related to each video game particularities, which in this case translates to a lower traffic load from the client side.

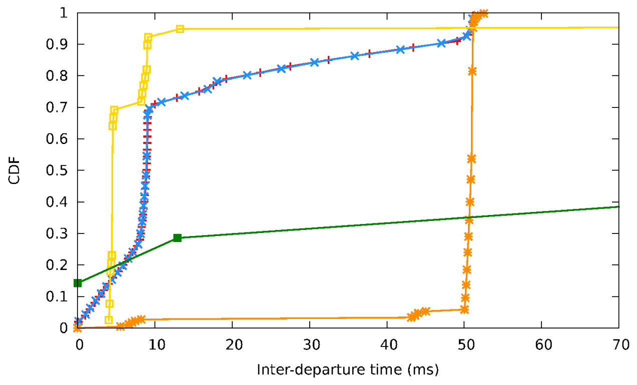
## 4.4 Upstream OnLive Traffic (Client → Server)

We have performed the same analysis for the client-originated traffic (*upstream*), and for the same to videgames (*crazytaxi* and *4elements*).

The CDF of packet size (in bytes) of *crazytaxi* is presented in Fig. 3a. It is interesting to note that the aggregated traffic is dominated by the Keys flow, since *crazytaxi* is a video game mainly played with the keyboard. This flow shows a maximum packet size of 300 bytes and a minimum of around 80 bytes. These packets, according to the CDF of inter-departure times (in ms) presented in Fig. 3b, are sent within a maximum period of 50 ms. Regarding the Mouse flow, it is shown that packets are sent in any case every 50 ms (see Fig. 3b), even though the mouse is not used during the playing session. In addition, Mic flow packets are also observed despite the fact that no voice chat was made. Finally, as observed in the figures, Control-ACK pack-



(a) CDF of packet size (bytes)



(b) CDF of packet inter-departure times (ms)

**Fig. 4** Upstream traffic characteristics of *4elements*

ets are mostly of about 90 bytes and are sent within a maximum of 10 ms period.

To end this study, the CDFs of packet size (in bytes) and inter-departure times (in ms) of *4elements* are presented in Fig. 4. In this case, the aggregated traffic is mostly related with the user keyboard's input (Keys flow), which presents similar features (of packet size and inter-departure times) to the flow observed in *crazytaxi*. This is due to the fact that *4elements* is a game also played with the keyboard, although some mouse interaction is also required. The rest of RTP flows show similar features to the ones shown by *crazytaxi*.

## 5 Modelling OnLive Traffic

Once the different OnLive flows have been fully characterised, we have separately modelled the traffic of *4elements* and *crazytaxi*. For each of the games we model both server-to-client traffic and client-to-server traffic. As previously stated, each of the UDP flows is comprised from several RTP flows. We model separate RTP flows, but calculate the goodness of fit of the model on the aggregated OnLive traffic. Aggregation is performed on flows of the same game and in the same direction (i.e., we separately depict goodness of fit for client-to-server traffic and server-to-client traffic). Besides the modelling methodology, this section also defines the parameters of the model and the goodness metrics for the validation of the resulting models, as well as the short description of the traffic generator implementation. In the end, we compare the parameters of empirical traces against the model, and versus the generated traffic in order to validate our model.

### 5.1 Modelling methodology

In this section we briefly present the methodology employed in the process of traffic modelling. We want to emphasize that, given the bursty nature of the OnLive traffic, we are not modelling individual packet sizes and packet inter-arrival times (IAT), but we model the size and inter-arrival times of bursts of packets sent within the RTP flows **XXX How are APDUs inferred from the traces?**. These can be considered as Application Protocol Data Units (APDUs), and can be smaller or larger than the Maximum Transmission Unit (MTU) of the underlying network technology. In this way the model retains the bursty properties of the traffic, and is independent of the underlying network technology. Also, it should be noted that the size of the RTP headers are included in the APDU size of the model.

Due to the relatively small packet rates and packet sizes we decided to ignore some of the RTP flows in the modeling process as their contribution to the overall traffic is negligible. For the server-to-client traffic we model the following flows: CBR-Audio, Cursor, VBR-Audio, and Video, while we ignore Monitor, Control and Chat. Regarding client-to-server traffic we model the flows Keys and Mouse in the case of *4elements*, and only Keys in the case of *crazytaxi*. We do not consider the cursor movement in the case of *crazytaxi* due to the fact that this game is only played with the keyboard, while the mouse is required for some actions in *4elements*.

We perform traffic modeling for each of the RTP flows by following the approach introduced by Paxson [30]. The algorithm is also described in detail in [21]. In this paper we only present it very briefly:

1. The probability distribution of the data set is examined and an appropriate analytical distribution is chosen. This is usually done through the visual examination of the Probability Density Function (PDF) or Cumulative Distribution Function (CDF) of the data.

**Table 5** Network traffic model parameters for *crazytaxi*

| Direction | RTP SSRC | Acronym | Count | APDU Model | APDU Parameters | IAT Model | IAT parameters |
|---|---|---|---|---|---|---|---|
| Downstream | 0x30000 | CBR-Audio | 14710 | Deter. | a=232 bytes | Deter. | a=10 ms |
| | 0x40000 | Cursor | 3156 | Deter. | a=44 bytes | Deter. | a=50 ms |
| | 0x50000 | VBR-Audio | 2460 | Deter. | a=1400 bytes | Deter. | a=50 ms |
| | 0x60000 | Video | 95718 | Deter. p=5.42%<br>Normal p=94.58% | a=298 bytes<br>$\mu$ = 12722, $\sigma$ = 3728 | Deter. | a=16 ms |
| Upstream | 0x0XXXX | Keys | 5107 | Deter. p=33.62%<br>Deter. p=3278%<br>Normal p= 33.60% | a=56 bytes<br>a=184 bytes<br>$\mu$ = 160.6, $\sigma$ = 66.13 | Uniform p=67.69%<br>Deter. p=32.31% | a=(0-49) ms<br>a=50 ms |

**Table 6** Network traffic model parameters for *4elements*

| Direction | RTP SSRC | Acronym | Count | APDU Model | APDU Parameters | IAT Model | IAT parameters |
|---|---|---|---|---|---|---|---|
| Downstream | 0x30000 | CBR-Audio | 20667 | Deter. | a=232 bytes | Deter. | a=10 ms |
| | 0x40000 | Cursor | 4456 | Deter. | a=44 bytes | Deter. | a=50 ms |
| | 0x50000 | VBR-Audio | 4486 | Deter. p=5.65%<br>Deter. p=66.25%<br>Normal truncated at 1400=28.1% | a=260 bytes<br>a=1400 bytes<br>$\mu$ = 1131.5, $\sigma$ = 3799 | Deter. | a=50 ms |
| | 0x60000 | Video | 75506 | Lognormal | $\alpha$ = 8.09, $\mu$ = 0.82 | Deter. | a=16 ms |
| Upstream | 0x0XXXX | Keys | 9302 | Deter. p=14.11%<br>Deter. p=4.58%<br>Deter. p=5.69%<br>Deter. p=11.01%<br>Deter. p=64.61% | a=56 bytes<br>a=60 bytes<br>a=68 bytes<br>a=80 bytes<br>a= 88 bytes | Deter. p=70.98%<br>Weibull p=29.02% | a=9 ms<br>$\gamma$ = 0.78, $\alpha$ = 44.47 |
| | 0x1XXXX | Cursor | 3943 | Deter. p=3.49<br>Deter. p=5.03%<br>Deter. p=67.38%<br>Deter. p=9.77%<br>Deter. p=7.22%<br>Deter. p=7.11% | a=152 bytes<br>a=168 bytes<br>a=184 bytes<br>a=200 bytes<br>a=216 bytes<br>a=232 bytes | Deter. | a=50 ms |

2. Graphical methods for comparison of distributions such as Quantile-Quantile plot (Q-Q plot) or Probability Plot (P-P plot) are used. Q-Q plot compares two distributions by plotting their quantiles against each other, while P-P plots two cumulative distribution functions against each other. If the resulting points on the plots are in a straight line, it means that the distributions are identical, but in practice usually there are deviations in the fit. By using these plots, it is easy to observe where deviations occur (e.g., lower tail, the main body, upper tail), and in these cases is maybe necessary to model the dataset with a split distribution.
3. Fitting the data set onto an analytical distribution (or more if needed). We used the *Minitab*[10] tool to determine the parameters of the distribution. Minitab implements both the maximum likelihood estimates (MLE) and the least squares (LSXY) methods, from which we use the parameters of the LSXY method.
4. Calculating the $\lambda^2$ discrepancy measure. As the standard goodness of fit tests are biased for large and messy datasets, a discrepancy measure is used [31]. We do not report the discrepancy's calculation procedure here, but an interested reader can find it in several works [30,21,37].
5. Examination of the tail, in search for deviations using the following expression:

$$\xi = log_2 \frac{a}{b} \tag{1}$$

   Where $a$ is the number of instances predicted to lay in a given tail, and $b$ is the number of instances that actually lay in that tail. If $b$ equals zero, it is replaced by 0.5. If the values of $\xi$ are positive, it suggests that the model overestimates the tail, and negative values indicate that the tail is underestimated.
6. Calculation of the autocorrelation function of the trace on which the modelling is based. Usually, short-term autocorrelation, or the autocorrelation at lag **XXX what does this lag mean?** 1 is reported.

### 5.2 OnLive Traffic Model

In this section we present the obtained models for the OnLive RTP flows of interest for *4elements* and *crazytaxi*. The reported values are in bytes and milliseconds and are presented in Table 5 and Table 6. It should be noted that Lognormal distribution is described with scale and location parameters. Each table comprises: (1) the traffic direction; (2) SSRC of the particular modeled stream; (3) the acronym of the flow; (4) the size of the dataset (i.e., number of units on which the modeling procedure has been done); (5) the model of APDU

[10] http://www.minitab.com/

sizes and IATs and the probabilities of specific distribution if a split modelling is applied; and (6) the values of the model parameters.

First we can notice that server's CBR-Audio and Cursor flows are game-independent. Almost all flows have constant packet rates (i.e., a fixed inter-arrival time). Only the user input (Keys flow) shows a significant of dispersion of the inter-arrival times. The most significant differences between games are in server-side flows *0x50000* and *0x60000*. As it is previously noted, *0x60000* carries the video so the differences in APDU sizes are related to the different games. On the other hand, *0x50000*, which has been previously identified as a VBR-Audio flow, also varies. As for the client side, it is interesting to note that *crazytaxi* has a wide array of values for packet sizes sent from the client. This can be attributed to the driving nature of the game because a continuous set of keystrokes are sent by the client (in *4elements* players can only swap positions of diamonds in the mesh).

As previously stated, we have modelled specific RTP flows, but examined how good is the model by calculating the discrepancy measure on the level of UDP aggregated traffic. The computation of the discrepancy for all aggregated traffic is simple for the APDU sizes, as this metric is additive for the calculation, since the values are always the same and have no impact on each other (i.e., one flow does not change the message size of the others)**XXX Clarify avoiding non-interaction**.

On the other hand, values of the IAT for aggregated traffic are dependent of interaction between different flows. In other words, if we started all flows at the same time we would create extremely bursty traffic due to the fixed parameters of the model. A high number of packets would be sent and received almost at the same time, resulting in a large increase of the near zero values in the inter-arrival time of the aggregated traffic. To alleviate this problem we examine the IATs of the aggregated traffic and vary the starting time of our RTP flows. To achieve this, we have found that the best fit for the IAT of the aggregated server traffic is a Weibull distribution. Then we iterate different starting times of our flows, and fit the resulting aggregated IATs to such Weibull distribution. We achieved the best fit for following starting times: CBR-Audio - 0 s, Cursor - 1 s, VBR-Audio - 3 s, Video - 0 s.

In Table 7 and Table 8 we present the goodness of fit of our models. The analysis is presented on the aggregated traffic for each game and for each direction. For the numerical description of the goodness of fit we use the discrepancy metric. In these tables we also present the data regarding tails of the distributions, particularly the number of observation in the tail, the percentage of observations in the tail in comparison with the whole dataset (as the value of the $\xi$ parameter), and whether the model underestimates the tail "-" and "0"(which appears in case of models which do not use continuous distributions) or overestimates it "+". Additionally we report the autocorrelation at lag 1 of the empirical dataset on which modelling is based. From the data in Tables 7 and 8 it is evident that the models of APDU sizes in general fit better than those of the IATs, although the best fit is for the upstream IAT of *4elements*. The amount of data observed in tails is negligible except for the cases of downstream IATs. The values of $\xi$ indicate that the model under/over estimation of the empirical data is not severe. In general correlation values are very low. To conclude, it should be noted that the mathematical description of the models' goodness of fit illustrates that models presented are good fits, with quality in line with the similar models found in the literature (e.g., [12]).

## 5.3 Experimental validation

For the experimental validation of the OnLive model we have used our previously developed software architecture for User Behaviour Based Network Traffic Generation (UrBBaN-Gen) [37]. UrBBaN-Gen is able to control the traffic generation process based on the player behaviour models. As we have not obtained the behavioural data of several players on the application level (e.g., how often each game is played and for how long), we simply set the behavioral parameters of UrBBaN-Gen to generate 1 flow of each game for the whole experiment duration. For traffic generation in UrBBaN-Gen we use the Distributed Internet Traffic Generator (D-ITG) developed on the University of Naples [1].

In order to use UrBBaN-Gen to generate traffic of OnLive games, we had to implement the network model of each flow in D-ITG. As some models are simple, their traffic can be generated from the command line interface of D-ITG. Other more complex models require the implementation of application-level models within D-ITG. We generate the traffic on a Linux Ubuntu 12.04 run in a virtual machine hosted by a PC with a i7@2.3GHz CPU and 4GB of RAM. Traffic was generated and captured on the loopback interface. Due to limitations in the traffic generator, each RTP flow was generated as a separate UDP flow.

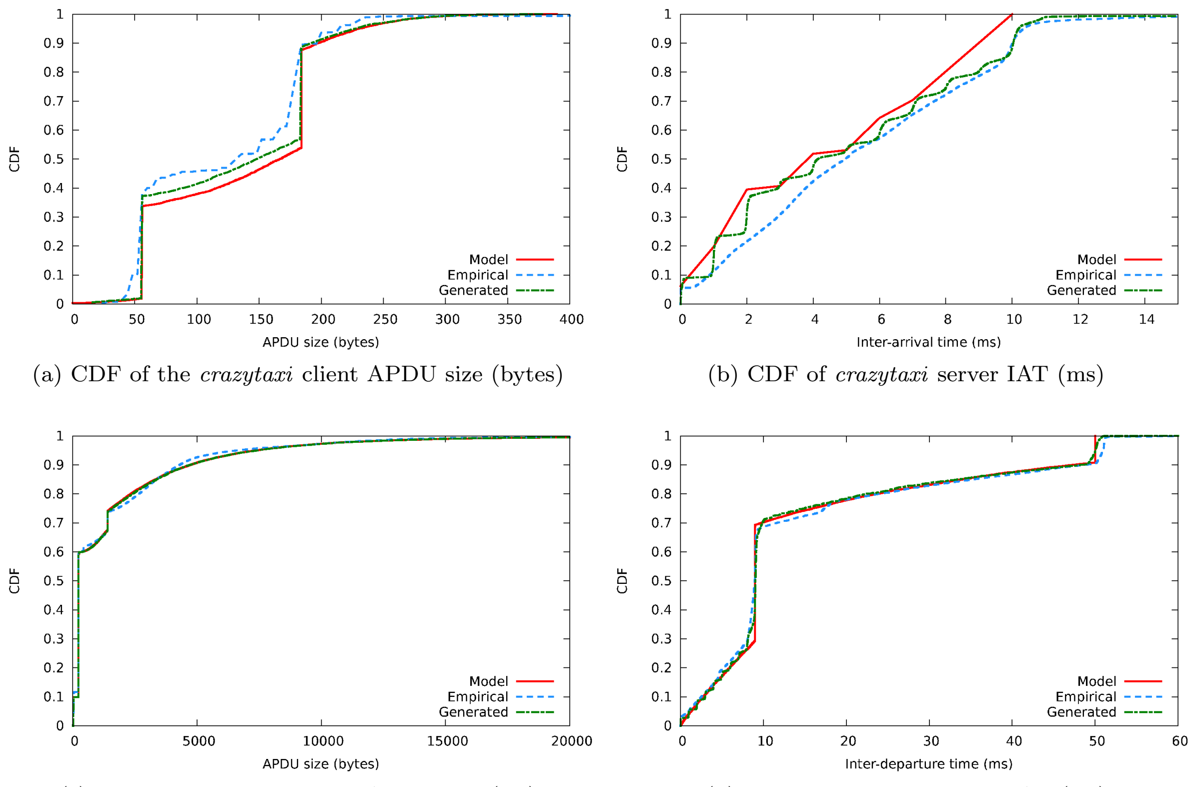
In Fig. 5 we present the results of our modelling procedures. Figure 5 comprises four subfigures. Each subfigure depicts one of the four modelled parameters: client APDU size, client IAT, server APDU size and server IAT. Two sub figures are based on the data from *4elements* and other two on *crazytaxi*. Each of the

**Table 7** Goodness of fit for traffic models of *4elements*

| Direction | Parameter | Discrepancy | Tail | Tail% | $\xi$ | Estimation | ACF |
|---|---|---|---|---|---|---|---|
| Downstream | APDU size | 0.07532 | 138 | 0.31981 | 0.13428 | + | -0.16737 |
| Downstream | IAT | 0.51748 | 1126 | 2.6095 | 0 | 0 | -0.00112 |
| Upstream | APDU size | 0.02721 | 77 | 0.17845 | 0 | 0 | 0.00912 |
| Upstream | IAT | 0.00705 | 12 | 0.02032 | 0 | 0 | 0.11913 |

**Table 8** Goodness of fit for traffic models of *crazytaxi*

| Direction | Parameter | Discrepancy | Tail | Tail% | $\xi$ | Estimation | ACF |
|---|---|---|---|---|---|---|---|
| Downstream | APDU size | 0.13322 | 78 | 0.2668 | -3.11548 | - | -0.33678 |
| Downstream | IAT | 0.65012 | 762 | 2.60664 | 0 | 0 | -0.33567 |
| Upstream | APDU size | 0.31903 | 29 | 0.42485 | 2.22948 | + | -0.0063 |
| Upstream | IAT | 0.13047 | 24 | 0.35185 | 0 | 0 | 0.01755 |



(a) CDF of the *crazytaxi* client APDU size (bytes)
(b) CDF of *crazytaxi* server IAT (ms)
(c) CDF of *4elements* server APDU size (ms)
(d) CDF of *4elements* client IAT (ms)

**Fig. 5** CDFs of PS and IAT of OnLive traces, the theoretical model and data extracted from the generated traffic.

subfigures contains three CDFs, which are the OnLive traces, the theoretical model, and the data extracted from the generated traffic. In this way we can observe how good does the model fit the captured traffic, wether the generated traffic significantly varies with respect to the model, and how well does the generated traffic describe the captured one **XXX What about the rest of figures?**.

The models closely follow the captured traffic for each of the parameters. Slight discrepancies can be observed on the client packets size of *crazytaxi*, as well as on the server IAT. Furthermore, our calculations of the discrepancy measure are confirmed as the models which have lower discrepancy values in Tables 7 and 8 have CDFs of the model and OnLive captured traffic almost coinciding in Figure 5.

As it can be observed, the parameters of the generated traffic closely follow the parameters of the model. Therefore, we deduce that the generation of the traffic creates no significant degradation of the imposed distributions. It is important to observe that the generation of the server traffic does introduce a small amount of jitter, and that visibly "altered" the CDF of the server IATs, leading to more similarity with the empirical traffic. In the end, we can confirm that the generated traffic describes the captured traffic accurately, as the CDF curves on most of the figures overlap almost completely.

## 6 Summary and Future work

In this paper we have studied the network traffic of the well-known OnLive cloud gaming platform. We have first conducted a reverse engineering of the OnLive protocol. To do so, extensive traffic traces from several video games of different genres have been collected and thoroughly analysed. It has been observed that an OnLive gaming session is divided into three phases: (a) authentication and global load balancing; (b) the OnLive main menu; and (c) playing a game. Moreover, we have fully characterised the traffic regarding the last phase (being it the most important one in terms of traffic load) of two video games (*crazytaxi* and *4elements*), which have the most different traffic profiles. Furthermore, we have proposed a per flow traffic model of OnLive for the same two video games. Finally, to demonstrate the correctness of our proposal, we have validated the generated traffic comparing it with the aggregated captured one, and we can conclude that our model describes accurately the OnLive traffic.

As further work we plan to conduct a reverse engineering of the Gaikai protocol. In addition, it could be interesting to compare these cloud gaming platforms taking into account different network scenarios (e.g. wired/mobile, bandwidth, delay, jitter, etc.).

**Acknowledgements** This work is partly supported by the projects: TEC 2012-32336, TEC 2010-12250-E and S-2009/TIC-1468; and by the Generalitat de Catalunya through the research support program project SGR-1202 and AGAUR FI-DGR 2012 grant. Also, this work was supported by the research project 036-0362027-1639 "Content Delivery and Mobility of Users and Services in New Generation Networks", by the Ministry of Science, Education, and Sports of the Republic of Croatia, and Ericsson Nikola Tesla, Zagreb, Croatia. Also, the research leading to these results has received funding from the European Community's Seventh Framework Programme under grant agreement no. 285939 (ACROSS).

## References

1. Avallone, S., Emma, D., Pescapé, A., Ventre, G.: Performance evaluation of an open distributed platform for realistic traffic generation. Perform. Eval. **60**(1-4), 359–392 (2005)
2. Borella, M.S.: Source models of network game traffic. Computer Communications **23**(4), 403–410 (2000)
3. But, J., Armitage, G., Stewart, L.: Outsourcing automated QoS control of home routers for a better online game experience. Communications Magazine **46**(12), 64–70 (2008)
4. Cevizci, I., Erol, M., Oktug, S.F.: Analysis of multi-player online game traffic based on self-similarity. In: Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games (2006)
5. Chen, K.T., Chang, Y.C., Tseng, P.H., Huang, C.Y., Lei, C.L.: Measuring the latency of cloud gaming systems. In: Proceedings of ACM Multimedia (2011)
6. Cheng, L., Bhushan, A., Pajarola, R., Zarki, M.E.: Realtime 3D Graphics Streaming using MPEG-4. In: Proceedings of the IEEE/ACM Workshop on Broadband Wireless Services and Applications (2004)
7. Choy, S., Wong, B., Simon, G., Rosenberg, C.: The brewing storm in cloud gaming: A measurement study on cloud to end-user latency. In: Proceedings of the 11th ACM/IEEE Annual Workshop on Network and Systems Support for Games (2012)
8. Cisco Systems, Inc.: Cisco Visual Networking Index: Forecast and Methodology, 2011-2016 (White paper) (2012)
9. Claypool, M., Claypool, K.T.: Latency and player actions in online games. Communications ACM **49**(11), 40–45 (2006)
10. Claypool, M., Finkel, D., Grant, A., Solano, M.: Thin to Win? Network Performance Analysis of the OnLive Thin Client Game System. In: Proceedings of the 11th ACM/IEEE Annual Workshop on Network and Systems Support for Games (2012)
11. Cricenti, A., Branch, P.: ARMA(1,1) modeling of Quake4 server to client game traffic. In: NetGames '07: Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games, pp. 70–74 (2007)
12. Dainotti, A., Pescapé, A., Ventre, G.: A packet-level traffic model of Starcraft. In: HOT-P2P '05: Proceedings of the Second International Workshop on Hot Topics in Peer-to-Peer Systems, pp. 33–42 (2005)
13. DFC Intelligence: http://www.dfcint.com/wp/?p=277. [Online; accessed 27-March-2013]
14. Färber, J.: Network game traffic modelling. In: NetGames '02: Proceedings of the 1st workshop on Network and system support for games, pp. 53–57 (2002)
15. Feng, W.c., Chang, F., Feng, W.c., Walpole, J.: A traffic characterization of popular on-line games. IEEE/ACM Transactions on Networking **13**(3), 488–500 (2005)
16. Goldmann, M., Kreitz, G.: Measurements on the Spotify peer-assisted music-on-demand streaming system. In: Proceedings of IEEE International Conference on Peer-to-Peer Computing, pp. 206–211 (2011)
17. Humphreys, G., Houston, M., Ng, R., Frank, R., Ahern, S., Kirchner, P.D., Klosowski, J.T.: Chromium: A Stream-Processing Framework for Interactive Rendering on Clusters. ACM Transactions on graphics **21**, 693–702 (2002)
18. Karachristos, T., Apostolatos, D., Metafas, D.: A realtime streaming games-on-demand system. In: Proceedings of the 3rd International Conference on Digital Interactive Media in Entertainment and Arts, pp. 51–56 (2008)
19. Kim, J., Hong, E., Choi, J.: Measurement and Analysis of a Massively Multiplayer Online Role Playing Game Traffic. In: Proceedings of Advanced Network Conference, pp. 1–8 (2003)
20. Kim, S., Kim, K., Won, J.: Multi-view rendering approach for cloud-based gaming services. In: Proceedings

of the 3rd International Conference on Advances in Future Internet, p. 102107 (2011)
21. Lakkakorpi, J., Heiner, A., Ruutuc, J.: Measurement and characterization of Internet gaming traffic. Tech. rep., Espoo (2002). Research Seminar on Networking
22. Lang, T., Armitage, G.: An ns2 model for the Xbox system link game Halo. Tech. Rep. 030613A, Swinburne University of Technology, Faculty of Information and Communication Technologies, Center for Advanced Internet Architectures (2003)
23. Lang, T., Armitage, G., Branch, P., Choo, H.Y.: A synthetic traffic model for Half-Life. In: in Australian Telecommunications, Networks and Applications Conference (2003)
24. Lang, T., Branch, P., Armitage, G.: A synthetic traffic model for Quake3. In: ACE '04: Proceedings of the 2004 ACM SIGCHI International Conference on Advances in computer entertainment technology, pp. 233–238 (2004)
25. Lee, Y.T., Chen, K.T., Su, H.I., Lei, C.L.: Are all games equally cloud-gaming-friendly? an electromyographic approach. In: Proceedings of the 11th IEEE/ACM Annual Workshop on Network and Systems Support for Games (2012)
26. Manzano, M., Hernandez, J.A., Urueña, M., Calle, E.: An empirical study of Cloud Gaming. In: Proceedings of the 11th ACM/IEEE Annual Workshop on Network and Systems Support for Games (2012)
27. Odlyzko, A.M.: Internet traffic growth: Sources and implications. In: Proceedings of SPIE, pp. 1–15 (2003)
28. Onlive: http://www.onlive.com. [Online; accessed 27-February-2013]
29. Park, H., Kim, T., Kim, S.: Network traffic analysis and modeling for games. In: Internet and Network Economics, Lecture Notes in Computer Science, pp. 1056–1065. Springer Berlin / Heidelberg (2005)
30. Paxson, V.: Empirically-Derived Analytic Models of Wide-Area TCP Connections. IEEE/ACM Transactions on Networking **2**(2), 316–336 (1994)
31. Pederson, S.P., Johnson, M.E.: Estimating model discrepancy. Technometrics **32**(3), 305–314 (1990)
32. Piri, E., Hirvonen, M., Laulajainen, J.P.: Empirical Evaluation of Streamed Online Gaming over WiMAX. In: Proceedings of the 8th International ICST Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities, pp. 255–270 (2012)
33. Ratti, S., Hariri, B., Shirmohammadi, S.: A Survey of First-Person Shooter Gaming Traffic on the Internet. IEEE Internet Computing **14**(5), 60–69 (2010)
34. Sandvine: Global Internet Phenomena Report. Tech. rep. (2013)
35. Shin, K., Kim, J., Sohn, K., Park, C.J., Choi, S.: Transformation Approach to Model Online Gaming Traffic. ETRI Journal **33**(2), 219–229 (2011)
36. Suznjevic, M., Dobrijevic, O., Matijasevic, M.: MMORPG player actions: Network performance, session patterns and latency requirements analysis. Multimedia Tools and Applications **45**(1-3), 191–241 (2009)
37. Suznjevic, M., Stupar, I., Matijasevic, M.: A model and software architecture for MMORPG traffic generation based on player behavior. Multimedia Systems (2012)
38. Svoboda, P., Karner, W., Rupp, M.: Traffic analysis and modeling for World of Warcraft. In: Communications, 2007. ICC '07. IEEE International Conference on, pp. 1612–1617 (2007)
39. The Telegraph: http://www.telegraph.co.uk/technology/. [Online; accessed 27-March-2013]
40. Wang, X., Kim, H., Vasilakos, A.V., Kwon, T.T., Choi, Y., Choi, S., Jang, H.: Measurement and Analysis of World of Warcraft in Mobile WiMAX networks. In: Proceedings of the 8th Workshop on Network and System Support for Games, p. 6 (2009)
41. Winter, D.D., Simoens, P., Deboosere, L., Turck, F.D., Moreau, J., Dhoedt, B., Demeester, P.: A hybrid thin-client protocol for multimedia streaming and interactive gaming applications. In: Proceedings of the international workshop on Network and Operating Systems Support for Digital Audio and Video, p. 15 (2006)
42. Wu, Y., Huang, H., Zhang, D.: Traffic Modeling for Massive Multiplayer On-line Role Playing Game (MMORPG) in GPRS Access Network. In: Proceedings of the International Conference on Communications, Circuits and Systems Proceedings, pp. 1811–1815 (2006)
43. Yoo, C.S.: Cloud Computing: Architectural and Policy Implications. Review of Industrial Organization **38**(4), 405–421 (2011)