Multimedia Systems (2014) 20:451–470
DOI 10.1007/s00530-014-0370-4

SPECIAL ISSUE PAPER

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

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

Published online: 28 March 2014

Abstract *Cloud gaming* is a new paradigm that is envisaged to play a pivotal role in the video game industry in forthcoming years. Cloud gaming, or gaming on demand, is a type of online gaming that allows on-demand streaming of game content onto non-specialised devices (e.g. PC, smart TV, etc.). This approach requires no downloads or game installation because the actual game is executed 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. As cloud gaming presents new challenges for both network engineers and the research community, both groups need to be fully conversant with these new cloud gaming platforms. The purpose of this paper is to investigate OnLive, one of the most popular cloud gaming platforms. Our key 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.

Keywords Cloud gaming · Online games · OnLive · Protocol · Reverse engineering · Traffic modelling

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

E. Calle
e-mail: eusebi@eia.udg.edu

M. Urueña · J. A. Hernández
Department of Telematic Engineering, Universidad Carlos III de Madrid, Madrid, Spain
e-mail: muruenya@it.uc3m.es

J. A. Hernández
e-mail: jahguti@it.uc3m.es

M. Sužnjević · M. Matijasevic
Faculty of Electrical Engineering and Computing, University of Zagreb, Zagreb, Croatia
e-mail: mirko.suznjevic@fer.hr

M. Matijasevic
e-mail: maja.matijasevic@fer.hr

1 Introduction

Although these days the traffic generated by networked gaming (i.e. by video game platforms such as PC, Xbox 360, PlayStation 3, and Nintendo Wii) only represents 0.03 % (77 PB) of total Internet traffic, it is expected to experience a 52 % compounded annual growth rate (CAGR) by 2016 [9], and become the Internet's traffic segment with the greatest expansion in the upcoming future. Online gaming brings players from all over the world together for fun and entertainment, and is regarded as one of the most profitable and popular Internet services. As a consequence, the video games business is expected to increase from $60.4 billion in 2009 to $70.1 billion in 2015 [14].

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

Many types of services taking advantage of the cloud have appeared in recent years: cloud storage (e.g. Dropbox, Google Drive), music-on-demand (e.g. Spotify, Pandora), video-on-demand (e.g. Youzee, Netflix), document editing and spreadsheets (e.g. Google Docs) and, more recently, cloud gaming (e.g. OnLive, Gaikai).

From a network perspective, the cloud paradigm has a direct impact on network load [47]. The traffic requirement of cloud games in comparison to standard games may increase by two orders of magnitude. To illustrate this point, World of Warcraft, a popular online game, has a bandwidth requirement in the vicinity of 55 kbps in the most demanding situations [40], while OnLive can require up to 5.6 Mbps [29]. While cloud computing reduces client hardware requirements, it may significantly increase network requirements to secure good quality of experience (QoE) of said services, as many of them are traffic-intensive. In fact, some business models like Spotify greatly rely on the performance of the network [17]: *premium* users require higher network capabilities than *free* users because they receive higher quality streams.

In this paper, we focus on the OnLive cloud gaming platform. At the time of writing, there are only a few cloud gaming service platforms available (among them OnLive, Gaikai, GamingAnywhere, GameString, G-cluster, Otoy, StreamMyGame, t5 labs, and iTSMY), with some of them still being under development. Interestingly, Gaikai was recently purchased by Sony for USD $380 million [43], and the mere fact that some of the most influential video game companies are now investing in cloud gaming would suggest that this new technology is positioned to take a considerable market share of the video game industry. In theory, if all of the "traditional" networked games were to be replaced with cloud-based games, game-related traffic would take second place in the overall traffic load of the Internet: behind Internet video, but surpassing the file sharing and web-related traffic categories.

In this light, network providers must be *au fait* with this new service which is one of the most demanding real-time services, both in bandwidth requirements (as we will show in following sections) and in delay requirements. This is because there is no computation of the virtual world state on the client side, or are there mechanisms for dealing with increased latency or packet loss on the client-side, or the so-called client side prediction (e.g. dead reckoning). Therefore, network requirements of cloud games are even greater than those of "regular" games. Network providers will only be able to handle this new service by upgrading their infrastructures to provide access technologies capable of dealing with the stringent network requirements of cloud gaming (as well as other cloud-based services), because current access technologies (i.e. 802.11g, ADSL, DOCSIS) may no longer be suitable to deliver high-quality cloud services. Additionally, to enhance the QoE of end users, network providers may prioritise cloud gaming flows over other non real-time flows through traffic management mechanisms, or even fine grain control, by prioritising specific flows (e.g. audio) within a cloud gaming service.

Traffic management systems require a comprehensive knowledge of traffic properties. Therefore, cloud gaming traffic, as a relatively new type of service, should be measured, and appropriate analyses should be made, so that the traffic properties are identified and understood. Furthermore, as deep packet inspection is very sensitive to privacy issues, and port and IP-based traffic classification is not that accurate anymore, traffic classifiers are relying heavily on the statistical properties of the traffic. Traffic models and synthetic traffic can be used to both design and test such traffic classification systems. In addition, analytical traffic models and traffic generators based on these are invaluable tools in both network operation and planning. Any network simulation, emulation or testing in both real and emulated environments is futile without good traffic models. The usability of traffic models in testing procedures is not just limited to network infrastructure, but is also valuable to the research community in the development and testing of new protocols, QoE studies, simulations, etc.

As different games may have very distinct game play mechanics and, therefore, traffic characteristics, traffic models need to be simple and general enough to encompass multiple games. Therefore, in this paper, we compare and model two games at the boundaries of the feasible traffic characteristics (i.e. games with the highest and lowest bandwidth requirement). Also, as cloud gaming traffic is very complex and comprises diverse types of information (e.g. user actions, video, audio, etc.) in the modelling process, it is very important to identify those traffic properties which are game dependent or not, or whether some flows have constant or variable bit rate, and so on. In this fashion, inspection of other games is significantly simplified as some traffic characteristics are only platform-dependent.

Previously mentioned methods should be applicable for cloud gaming services in general. Therefore, a comparison of different cloud gaming services is essential to obtain the characteristics which are not platform specific, but rather are common to the whole cloud gaming paradigm. For instance, when comparing the video flows of two cloud gaming providers, the first step is to understand the protocols employed and the different traffic patterns generated. As the protocols employed are most often proprietary, the only approach is to reverse engineer them. In this paper, we take a first step towards such a comparison of cloud gaming services by analysing and reverse-engineering the OnLive protocol.

In our previous work [29], we carried out a traffic measurement comparison between OnLive and Gaikai and concluded that there are several significant differences between the two platforms. However, to the best of our knowledge, there are no previous works focusing on the specific protocols used by these cloud gaming platforms. As a consequence, we have focused our work on conducting a reverse engineering study on OnLive, leaving Gaikai for future study.

This paper has two main objectives: (1) to study the OnLive protocol, based on extensive traffic traces of several games; and (2) to propose a general OnLive traffic model which can be used for network dimensioning, planning optimization and other studies.

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

2 Related works

Since its inception, the Internet has continuously evolved, nearly doubling every year [30]. In the 1990s, 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 services are becoming increasingly popular and are continuously expanding 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 [37].

In the case of video games, several studies have previously analysed the impact of online gaming on the network. In [16], the authors studied several popular online games. For instance, it was reported that Unreal Tournament 2003 could be played using 28.8 kbps modems. Furthermore, the authors of [4] studied the traffic load produced by the popular Call of Duty: Modern Warfare 2 game and reported that server-originated traffic produces only 34 kbps, while client traffic is about 13 kbps. Later studies analysed World of Warcraft, one of the world's most popular video games, and reported that it only requires a maximum 5 kbps uplink and a 55 kbps downlink [40], thus drawing the same conclusion: traditional online video gaming is not a bandwidth-hungry application at all.

Cloud computing has recently emerged as a new paradigm for hosting and delivering services over the Internet, and would appear to be a remarkable opportunity for the video game industry. The authors of [22] have defined four requirements for cloud-based gaming services:

- *User responsiveness*: to guarantee suitable responsiveness, interaction in the games should be kept below 80 ms [10].
- *High-quality video*: to provide high quality video, interactive video ($1,280 \times 720$ resolution), data should be reduced as much as possible while still maintaining quality.
- *Quality of service (QoS)*: to decrease latency and avoid network bottlenecks, game traffic should be prioritised. An example of such a technique was presented in [3].
- *Operating costs*: to minimize power consumption, optimization techniques must be developed.

In this same paper, the authors defined two approaches to 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 [19].
- *Video streaming*: the server renders the whole 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, 20]. This approach is most suitable for thin clients, because they lack hardware-accelerated rendering capabilities [45], and so is used by the majority of cloud gaming platforms.

In [35], the authors evaluated the quality of streamed online gaming over fixed WiMAX and concluded that it 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. However, the work went no 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 streamed as it is created. As such, no traditional compression techniques can be applied to the entire video and consequently, the server must compress the video in real-time to reduce the bandwidth requirements (i.e. with H.246), albeit increasing the latency.

Furthermore, a QoE analysis of three game genres was carried out in [28]: 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.

The work presented in [11] 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, as a service, OnLive is fundamentally different from Skype and YouTube (in terms of network performance metrics), and that it adapts to changes in network conditions (i.e. delay or bandwidth).

As previously mentioned, in [29], the authors analysed the traffic characteristics of OnLive and Gaikai. It was pointed out that both platforms were similar in their packet size distribution, but significantly different in their packet inter-arrival times (IAT). Furthermore, it was also observed that traditional online gaming is remarkably different from cloud gaming in terms of network load and traffic characteristics.

Finally, Shea et al. [39] analysed the main challenges to the widespread deployment of cloud gaming. In their work, the authors stated that OnLive uses a proprietary version of the real-time transport protocol (RTP) as well as a proprietary version of the H.264/MPEG-4 AV encoder.

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

Initially, most works in this area were focused on the First-Person shooter (FPS) genre and followed the modelling approach pioneered by Paxson [33]. 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 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 [27], the authors modelled the traffic of *Quake III*. In addition to modelling PS and IAT, the impact of different graphic cards, the number of players, and maps (i.e. virtual worlds where the game takes place) were investigated. The traffic of the game *Halo* by Microsoft for the Xbox console was also analysed and modelled [25] as well as *Half-Life* [26], where OpenGL versus software-based rendering and its impact on network traffic were examined. Färber [15] analysed traces of *Counter Strike*, which were gathered during a 36-h LAN party involving 37 players, and created a traffic model dependent on the number of active clients. A different kind of modelling approach was presented by Cricenti and Branch [12] who modelled the traffic of *Quake 4* using mixed autoregressive/moving average (ARMA) models.

In addition to FPS games, real-time strategy (RTS) games such as Starcraft [13], and Massive Multiplayer Online Role-Playing Games (MMORPGs) have also been studied as a result of the increase in their popularity among the player base. Svoboda et al. [42] analysed traffic traces captured within the 3G mobile core network and noted that WoW was among the top 10 TCP services in the monitored network and consumed 1 % of all TCP traffic. They also provided a traffic model for WoW. Wu et al. [46] modelled the traffic of one of the most popular MMORPGs in China—*World of Legend*, while Kim et al. [21] presented traffic models for *Lineage*.

As the traffic of MMORPGs is highly variable and difficult to model, new approaches to traffic modelling have appeared in the literature. Shin et al. developed novel transformational modelling procedures and compared them to standard modelling techniques. A further approach taken by several research groups was to classify user behaviour at application level and model separate behaviours [23]. All classification approaches [32, 41, 44] used WoW as their case study.

As cloud gaming is still a relatively new technology, there are still several aspects that need detailed investigation. To compare different services and extract the general characteristics of cloud gaming traffic independent of the platform, understanding the protocols employed by different cloud gaming providers is crucial. As these protocols are usually proprietary, they must be reverse engineered to be able to understand their procedures and the traffic patterns they generate. To the best of our knowledge, no reverse engineering of the protocols used by cloud gaming platforms or traffic models has yet been proposed. Therefore, in this paper, we carry out a reverse engineering of the OnLive protocol and propose a network traffic model for this platform. The main purpose is to provide the research community and network engineers with knowledge of the network load generated by these emerging applications, which, in turn, will help to design future network infrastructure.

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 such as LG and

Sony. According to OnLive's website [31], minimum requirements include a network connection with minimum bandwidth of 2 Mbps to render content at 1,024 × 576 resolution. For content at 1,280 × 720 resolution, a network connection of 5 Mbps or higher is recommended. OnLive requires a local installation of its own client software.

We have studied the behaviour of the OnLive protocol by analysing the packet traces previously employed to characterise the aggregated OnLive traffic [29]. On closer inspection, we were able to identify the different flows that compose such traffic, and the protocols employed to transport it. To identify the purpose of each flow, we have carried out additional experiments in the same scenario (described later in Sect. 4.1), where we have selectively filtered individual flows to observe the effects 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 used 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 phase, the *OnLive Client* authenticates (using a TLS/TCP connection) and measures the latency and available bandwidth with different OnLive sites. In the second phase, once a suitable OnLive site is selected by the client, the OnLive servers start 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 fashion, employing multiple RTP/UDP flows multiplexed over a single UDP port (the default OnLive UDP port is 16,384).

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

3.1 Phase 1: Authentication and global load balancing

The first phase of the OnLive protocol is shown in Fig. 1. 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`, using a TLS/TCP encrypted session (port 443). Therefore, we are unsure of the messages exchanged through such connection, although they are quite likely to be HTTP-based and involve authenticating the user, as well as probably obtaining the IP addresses of several OnLive servers that are employed in the latency measurements that follow.

At the time of this paper being written (March 2013), OnLive has servers in at least four different locations (i.e. London in Europe, and Washington, Dallas and San Jose in the USA). This allows OnLive to cover most of the USA and western Europe within the 1,000 mile radius which OnLive considers close enough to have acceptable round-trip time (RTT), although other studies would suggest otherwise [7]. Once authenticated, and given the importance of having a low RTT with the server for game responsiveness, the client starts measuring its RTT with those locations to find the most suitable OnLive site.

These active RTT measurements are performed using what would seem to be a custom probing protocol conveyed in short RTP/UDP messages, although its behaviour resembles other active measurement protocols such as [18, 38]. As observed in Fig. 1a, the client starts sending batches of nine RTP messages every 22 ms to 9 different IP addresses (UDP port 16,384) belonging to 2 or 3 *OnLive Probing Servers* in each of the four locations. Each RTP message comes from a different UDP port of the client and has a different synchronised source identifier (SSRC) value, but all share the 0x2800XXXX prefix which, as we will later see, 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, 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 to the client with similar RTP messages, changing the payload but keeping the same SSRC, Sequence Number and Timestamp as the client messages they are echoing. With this information, the client can easily measure the RTT with each server, as well as detect lost packets. After a fixed number (i.e. 12) of RTP exchange rounds from the first response from each server, the servers then send a final RTP message with a slightly different SSRC = 0x28YYXXXX that signals the end of the measurement session. As we do not have access to the source code of the OnLive client software, we do not know the details of the server selection algorithm but it appears to use the RTT as its main metric, because the client systematically selects one of the first responding servers, which are also the ones finishing the measurement session first.

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

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

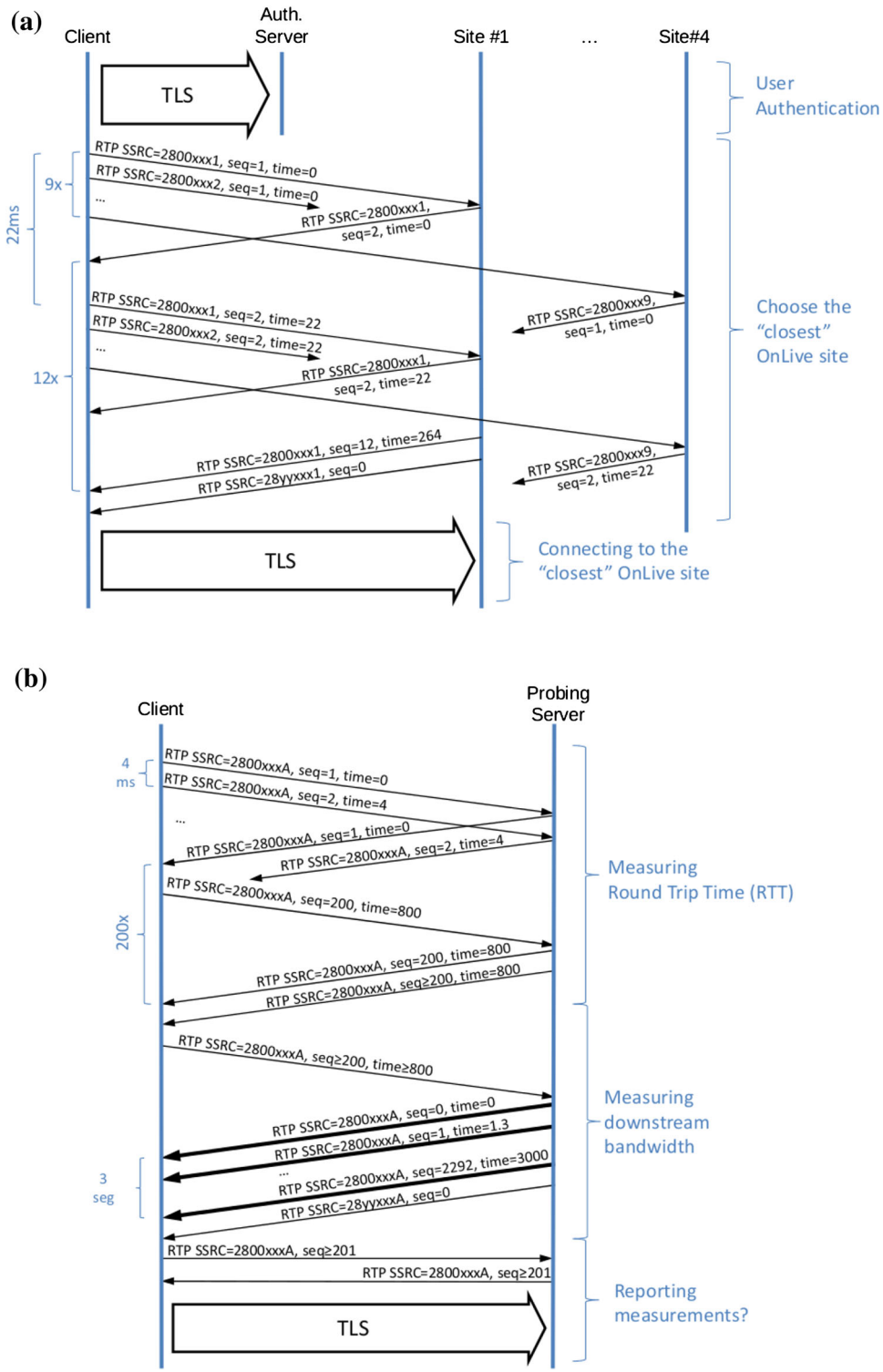


Fig. 1 Phase 1 of the OnLive protocol. **a** Phase 1a: choosing the "closest" OnLive site. **b** Phase 1b: measuring the RTT and downstream bandwidth

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 (as shown in Fig. 1b). This measurement session starts in a similar manner to the previous one, since it also employs an SSRC = 0x2800XXXX identifier (although different to the previous one) and the short RTP messages from the client are echoed by the

Table 1 RTP/UDP flows of the OnLive Streaming Protocol

Direction	RTP SSRC	RTP Payload Type	Short name	Alias
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)

server (16,384 port), but now the RTP messages are sent every 4 ms and last 200 rounds. Moreover, when this latency measurement session is over, the client sends an RTP message to a different UDP port[3] of the server to start an RTP flow to measure the client's downstream bandwidth.

Although this RTP flow maintains the same SSRC as the previous one, now these RTP messages are only sent by the OnLive Probing Server and are not replied by the client. These packets are quite large (1,400 bytes) and sent every 1.3 ms until the 2292 sequence number is reached (i.e. 3 s at 8.5 Mbps), when the server stops sending them. There is a final exchange of control RTP messages where the server notifies the client of its public IP address and port, probably to detect intermediate NATs. It is worth noting that all OnLive RTP flows and TCP connections are initiated by the client to avoid NAT and firewall issues.

Once these two active measurement sessions with the selected OnLive probing server are over, the client may now estimate both the RRT 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. [11].

The client probably then notifies the values of these measurements back to the OnLive Probing Server by means of the ongoing 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 subsequent phase of the OnLive session, which shows the main menu.

[3] This port is dynamic and larger than the OnLive's 16,384 default one. The port number is notified to the client using the custom protocol encapsulated in echo RTP messages from the server.

3.2 Phase 2: OnLive main menu

The main menu of OnLive is not a web page, or even a Flash Player animation, but rather 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 to 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 an 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 those employed by the OnLive Probing Protocol (i.e. SSRC = 0x28XXXXXXX). In particular, in this phase, there are several RTP flows multiplexed in the same UDP flow, both in the downstream (server-to-client) and upstream (client-to-server) directions. Table 1 shows all RTP flows which, for the sake of clarity, have been named as indicated in the *Short name* column. Figure 2 depicts the RTP subflows observed in this phase and, as shown, the different RTP flows can only be identified by their SSRC values. 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 96 and 101 Payload Types, 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

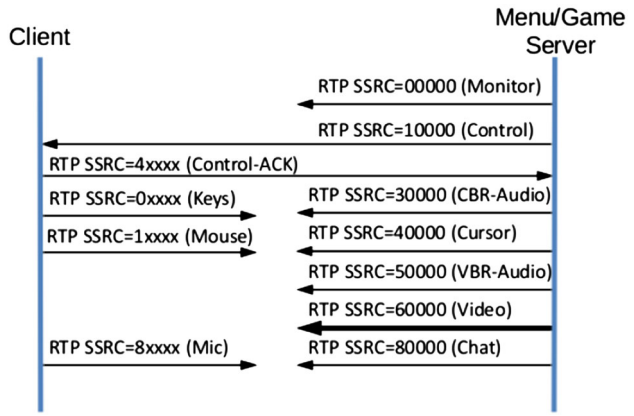


Fig. 2 Phases 2 and 3: OnLive RTP subflows

messages of the OnLive Streaming Protocol also include a Header Extension with three 32 bit-long fields that contain a microsecond timestamp and a common counter of all RTP messages generated by the source client/server, irrespective of the RTP flow they belong to. This timestamp field in the RTP extension header has allowed us to precisely measure the IAT of OnLive packets without the jitter generated by the network.

Although, as previously 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 explain only those flows related to control functions, while the game-related flows will be explained in the following subsection.

As shown in Fig. 2, 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 QoS. Whenever an OnLive menu or gaming session begins, this flow starts with a burst of four long RTP messages which are then followed by a small (i.e. 48 bytes) RTP message sent 10 ms later. This sequence is repeated several times, apparently to measure the available bandwidth with the new OnLive Menu Server (if it is too low—the OnLive website states 2 Mbps—the 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 RTT to the server (the client shows a warning when these packets are filtered).

The next RTP flows to appear are the Control and Control-ACK, sent by the server and the client, respectively. In fact, these two flows work together in what we have interpreted as some kind of control protocol. When the server wants to convey a message to the client it sends an RTP packet with SSRC = 0x10000 that the client then 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 makes a selection from the OnLive main or in-game menus, such as configuring or muting[4] the audio, which the client then has to be notified about.

The client notifies the servers about the actions of the user by means of the Keys flow (which will be described later as it is mostly employed in the gaming phase), as well as by means of the Mouse and Cursor flows, which are related to the mouse pointer. These two flows have a fixed rate of one RTP message every 50 ms regardless of the client's mouse movements. Moreover, the packets of both flows are interleaved and there is a 25 ms gap between a client and a server packet. 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 from the OnLive menu. At that moment, the OnLive client stops communicating with the OnLive Menu Server and opens a new UDP socket to send an RTP message to the *OnLive Gaming Server* which also uses the OnLive default 16,384 port. In most of the sessions captured, the Menu and the Gaming Servers have consecutive IP addresses, and in some cases they have the same IP address.

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 in Table 1 and Fig. 2. These are the most important flows of this phase and of the OnLive Streaming Protocol as such.

In particular, the Video flow has the highest rate because it carries the OnLive video stream. The RTP messages in 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 must employ a variable bit rate (VBR) video codec. The authors of [39] have stated that

[4] Interestingly, the muting audio option does not interrupt the audio flows, but simply notifies the client not to play them.

OnLive uses a version of the H.264/MPEG-4 AV encoder, but we have not been able to verify this.

The audio stream is not integrated into the video RTP flow, but rather is sent as a separate flow. In fact, 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 even when dropping any of them, unless both are filtered simultaneously). Furthermore, each flow not only seems to use a different codec, but even a different RTP encapsulation. The CBR-Audio messages have a 100 Payload Type value, the Timestamp field is set to zero, and they carry a fixed 204 byte payload that is sent every 10 ms. 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. It is possible that this flow is related to the 5.1 surround audio supported by some games, but we have not been able to verify this because we have not been able to identify the audio codecs.

The user actions, other than moving the mouse cursor as explained in the previous subsection, are conveyed to the server in the Keys flow, with payloads ranging from 20 to 200 bytes, and are sent on-demand when the user presses a key or mouse button.

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

The two remaining RTP flows in the OnLive Streaming Protocol are related to the Voice Chat function that allows OnLive users to talk amongst themselves. 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 the communication with the previous server (e.g. OnLive Gaming Server) and employs a new UDP port to communicate with the new server (e.g. OnLive Menu Server).

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

4 Traffic characteristics of OnLive

This section presents the measurement methodology followed to carry out the reverse engineering of the OnLive protocol and to study its traffic characteristics. This section is divided into three subsections. We first explain the measurement scenario and methodology, followed by an overview of the results obtained from the measurements carried out. Next, we focus on the last phase of the OnLive protocol (i.e. playing a game). We present both the server and client-originated traffic for two specific games, explicitly selected because they have shown the two most disparate traffic profiles [29].

4.1 Measurement methodology

To identify the main features of the traffic patterns, we captured traces for five different games served by the

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	149,004	174,265	13,867	153,798	108,619
Avg. packets/s	597.41	667.25	691.39	641.66	459.09
Avg. packet size (B)	915.57	975.05	1,014.99	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/s	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

OnLive platform. Table 2 summarises the chosen games, along with their nicknames and genres.

Traffic traces have been captured using Wireshark[5] at the local computer of the gamer. Measurements have been carried out from Spain to the nearest OnLive data centre (London) 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. The access network characteristics have been obtained using *pingtest*[6] (jitter) and *speedtest*[7] (downstream and upstream bandwidth), plus the *traceroute* command-line tool (RTT). In [29], the same traffic characteristics were also obtained from a commonplace wireless home connection (802.11n + VDSL); however, in this work we have only considered the former network scenario.

For each game, the capture of traffic traces was started at the time of executing the client program and then finalised after playing for 100 s. Since 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 IAT of all captured packets.

Thereafter, we thoroughly analysed the traffic traces 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*.[8]

4.2 Overview

Here, a detailed summary of the overall measurement results is presented in Table 3. The table shows the measured traffic metrics for each video game. Traffic metrics are separated into *downstream* and *upstream* directions. There is a striking difference between the bandwidth required by OnLive cloud gaming platform and "traditional" online games, such as Call of Duty (34 kbps in downstream and 13 kbps in upstream) [4]; World of Warcraft (which only reaches 55 kbps in downstream and 5 kbps in upstream) [40] and Unreal Tournament 2003 (which can be played with 28.8 kbps modems) [16, 36].

Next, the server and client-originated data traffic from *crazytaxi* and *4elements* is described, because these show the most different required bandwidth values (see Table 3). In both cases, we have considered the playing phase of the OnLive protocol because it is the most significant one in terms of traffic load.

[5] http://www.wireshark.org/.

[6] http://www.pingtest.net/.

[7] http://www.speedtest.net/.

[8] http://www.netfilter.org/projects/iptables.

Table 4 Summary of the characteristics of the RTP flows observed in the last phase of the OnLive protocol

	Avg. packet size (bytes)	Bit rate (kbps)
Downstream		
Aggregated	688.38/1017.61	2450/6020
Monitor	180.51/198.62	1/2
Control	96.60/98.88	14/1
CBR-Audio	274.00/274.00	200/200
Cursor	86.00/86.00	10/10
VBR-Audio	1250.25/1348.15	210/220
Video	786.69/1157.43	2020/5632
Chat	152.67/131.78	1/0.1
Upstream		
Aggregated	154.89/175.10	120/70
Keys	155.09/175.11	80/50
Mouse	233.24/221.65	40/20
Control-ACK	86.80/86.65	3/0.1
Mic	315.00/253.64	0.4/0.2

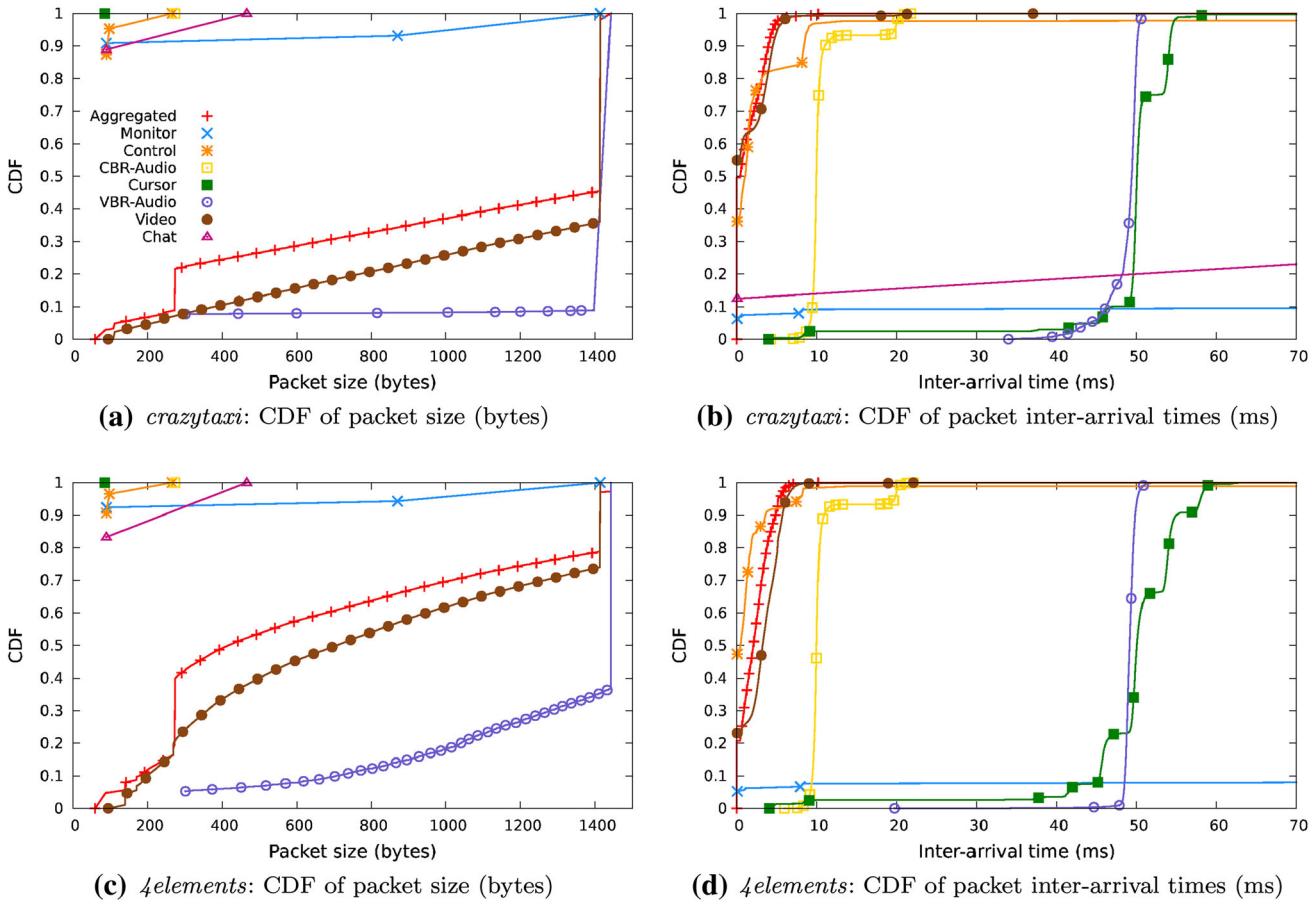
Values are displayed as follows: *4elements*/*crazytaxi*

Table 4 presents a dissection of the different RTP flows, which have been observed in the gaming phase of the OnLive protocol. As previously shown in Table 1, it can be ascertained that there are seven different RTP flows in the downstream, while there are just four in the upstream. It is worth highlighting that the CBR-Audio and Cursor flows have constant packet size and packet IAT. While VBR-Audio may be the flow composed of the largest packets, Video flow actually comprises the majority of the bandwidth used by OnLive.

Finally, it is also worth noting that the packets from the Video flow reach the client in bursts. Those consecutive Video flow packets have exactly the same RTP timestamp and the last packet of the burst usually has the RTP Mark flag set. Therefore, we think that these bursts carry a video frame that does not fit into a single RTP message. The average burst IAT for *4elements* and *crazytaxi* are 16 ms; the average number of packets per burst for these two games is around 5.4 and 11, respectively, and the average burst length (measured as the sum of RTP payloads) is around 3.8 and 12 kb, respectively.

4.3 Downstream OnLive traffic (server → client)

The measurements from the server-originated traffic (*downstream*) are analysed next. The cumulative distribution functions (CDF) of packet sizes and packet IAT for *crazytaxi* and *4elements* are shown in Fig. 3. Results for the aggregated traffic of the playing phase of the OnLive protocol are compared with all the particular RTP flows



(a) *crazytaxi*: CDF of packet size (bytes)

(b) *crazytaxi*: CDF of packet inter-arrival times (ms)

(c) *4elements*: CDF of packet size (bytes)

(d) *4elements*: CDF of packet inter-arrival times (ms)

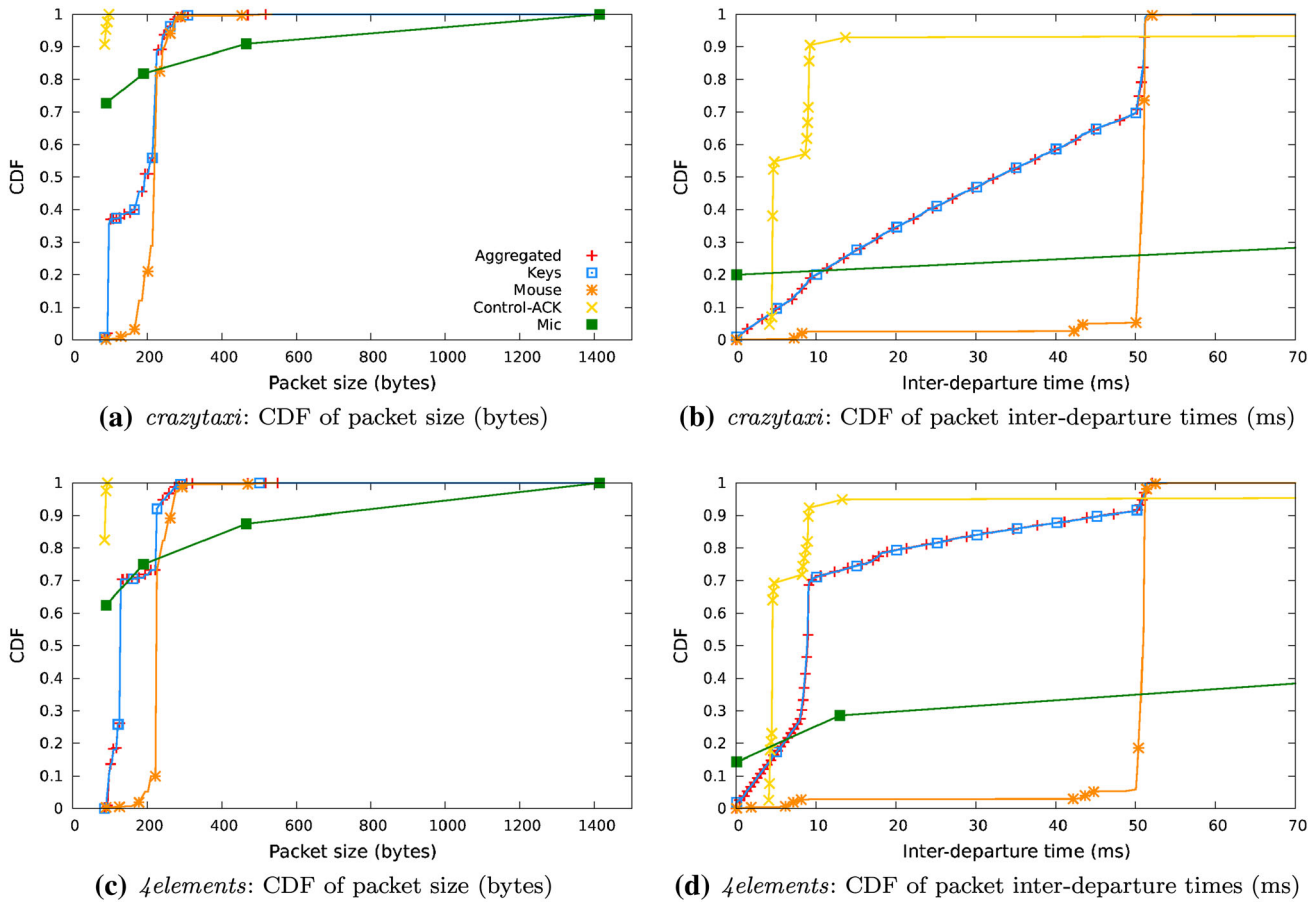
Fig. 3 Downstream traffic characteristics of *crazytaxi* (**a**, **b**) and *4elements* (**c**, **d**)

generated by OnLive in that phase. The aggregated values are similar to those reported in the literature [8, 11, 29], thus, we will now focus on the separate RTP flows.

Figure 3a depicts the CDF of packet size (in bytes) of *crazytaxi*. The aggregated curve is shown together with the dissected RTP flows. The Monitor flow line indicates that almost all packets in 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 Maximum Transmission Unit (MTU) size (i.e. 1,418), although no fragmentation occurs at the network layer. Furthermore, it can be observed that the Video flow plays a pivotal role regarding the CDF of the aggregated traffic. In that flow, while 65 % of packets are of maximum size, the other 35 % present a uniform distribution from 50 bytes to this maximum. Thus, as mentioned in Sect. 4.2, it can be inferred that the Video flow is composed of bursts, which cannot fit into a single RTP message, forcing the bursts to be split[9] into several packets. The Chat flow is mostly composed of small packets.

With regard to the IAT (in milliseconds) of *crazytaxi*, Fig. 3b shows several important aspects which help 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; the average inter-packet delay is higher than 70 ms (not shown in Fig. 3b for the sake of clarity). To the contrary, the CBR-Audio, VBR-Audio, and the Cursor flows are predominantly constant in their IAT; being around 10, 50 and 10 ms, respectively. It can also be observed that the OnLive Control flow packets are mostly sent within and up to a 10-ms period. Finally, it is important to note that the Video flow packets are the most frequent ones, the majority of them are being sent within and up to a 6-ms period.

[9] This video frame splitting is performed at the application layer, that is, several RTP messages are generated, instead of a single UDP packet being fragmented by the IP layer.



(a) *crazytaxi*: CDF of packet size (bytes)

(b) *crazytaxi*: CDF of packet inter-departure times (ms)

(c) *4elements*: CDF of packet size (bytes)

(d) *4elements*: CDF of packet inter-departure times (ms)

Fig. 4 Upstream traffic characteristics of *crazytaxi* (**a**, **b**) and *4elements* (**c**, **d**)

The CDF of the packet size of *4elements* is shown in Fig. 3c. It is worth mentioning that most RTP flows (Monitor, Control, CBR-Audio, Cursor and Chat) present very similar curves to those obtained for *crazytaxi*. However, as can be observed, VBR-Audio and Video are significantly different from the same RTP flows of *crazytaxi* and consequently vary the CDF of the aggregated traffic. While 90 % of the *crazytaxi* VBR-Audio packets are of maximum size, only around 60 % of the packets show this maximum size in the case of *4elements*. Moreover, the portion of maximum size packets from the *4elements* Video flow is about half that of *crazytaxi*. As a consequence, it can be inferred that the traffic generated by the VBR-Audio and the Video flows is dependent on the game that is being played. Thus, if other OnLive games were to be studied in future investigations, these two flows would be the most important ones to be taken into consideration.

Finally, Fig. 3d presents the CDF of IAT (in ms) of *4elements*. As in the case of *crazytaxi*, the Monitor and Chat flows of *4elements* are the least frequent ones (the latter does not appear in Fig. 3d because it has a too high value of IAT). In addition, CBR- and VBR-Audio flows and the Cursor flow are subjected to specific IAT: 10, 50 and 10 ms, respectively. It is interesting to note that for *4elements*, the VBR-Audio flow shows more IAT modes than for *crazytaxi*. The Control flow is roughly the same as that 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's idiosyncrasies, 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 two video-games (*crazytaxi* and *4elements*).

The CDF of the packet size (in bytes) of *crazytaxi* is presented in Fig. 4a. It is interesting to note that the aggregated traffic is dominated by the Keys flow, because *crazytaxi* is a video game played mainly using 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 presented in Fig. 4b, are sent within a maximum period of 50 ms. In the

case of the Mouse flow, it is demonstrated that packets are always sent every 50 ms (see Fig. 4b), even though the mouse is not used during the playing session. In addition, Mic flow packets are also observed despite the fact that there was no voice chat. Finally, as observed in the figures, Control-ACK packets are mostly of about 90 bytes and are sent within a maximum of 10-ms period.

To end this study, the CDFs of the packet size (in bytes) and inter-departure times (in ms) of *4elements* are presented in Fig. 4c and d, respectively. In this case, the aggregated traffic is mostly related to the user's keyboard 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 which is also played with the keyboard, although some mouse interaction is required as well. The rest of RTP flows show similar features to those 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 and client-to-server traffic. As previously stated, each of the UDP flows is composed of several RTP flows. We model separate RTP flows but, to save space, we only present the goodness of fit for the model of the aggregated OnLive traffic. Aggregation is performed on flows from the same game and of the same direction (i.e. we depict goodness of fit separately for client-to-server and server-to-client traffic). Besides the modelling methodology, this section also defines the parameters of the model and the goodness metrics for validating the resulting models, as well as providing a short description of the traffic generator implementation. At the end, we compare the empirical traces against the generated traffic to validate our model.

5.1 Modelling methodology

In this section, we briefly present the methodology employed in the process of traffic modelling. We present a model of individual packet sizes and packet IAT. In addition, we have modelled the size and IAT of bursts of packets sent within the Video RTP flow. These bursts are identified using their RTP timestamp value (i.e. payloads of all subsequent packets with the same RTP timestamp value are grouped into one burst). These can be considered as application protocol data units (APDUs).

It should be noted that the RTP headers are not included in the packet size (PS) of the model, so the model only comprises RTP payload traffic. When inspecting the packets obtained from the traffic traces, we first remove 58 bytes of headers from each packet (14 bytes in Ethernet, 20 bytes in IPv4, 8 bytes in UDP, and 16 bytes in RTP), and then perform modelling on the remaining payload.

Due to the relatively small packet rates and packet sizes we decided to ignore some of the RTP flows in the modelling 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. For client-to-server traffic, we model the Keys flow and ignore Mouse, Control-ACK, and Mic flows.

We perform traffic modelling for each of the RTP flows by following the approach introduced by Paxson [33]. The algorithm is described in detail in [24]. In this paper, we only present it very briefly:

1. Examination of the probability distribution of the data set and selection of an appropriate analytical distribution. This is usually done through the visual examination of the probability density function or cumulative distribution function (CDF) of the data.
2. Use of graphical methods for distribution comparisons such as Quantile–Quantile plot (Q–Q plot) or Probability Plot (P–P plot). Q–Q plot compares two distributions by plotting their quantiles against each other, while P–P plots compare two cumulative distribution functions against each other. If the resulting points on the plots are in a straight line, this means that the distributions are identical, but in practice there are usually deviations in the fit. Using these plots, it is easy to observe where deviations occur (e.g. lower tail, the main body, upper tail), and in these cases it may be necessary to model the dataset with a split distribution.
3. Fit the data set onto an analytical distribution. 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. Calculation of the λ^2 discrepancy measure. As the standard goodness of fit tests are biased for large and messy datasets, a discrepancy measure is used [34]. While we do not report the discrepancy's calculation procedure here, an interested reader can find it in several works [24, 33, 41].
5. Examination of the tail. We searched for deviations using the following expression:

$$\xi = \log_2 \frac{a}{b} \quad (1)$$

where a is the number of instances predicted to lie in a given tail, and b is the number of instances that

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

actually do lie in that tail. If b equals zero, it is replaced by 0.5. If the values of ξ are positive, it would suggest 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 1 is reported.

5.2 OnLive traffic model

In this section, we present the proposed models of packet sizes and packet IATs for the OnLive RTP flows of interest for *4elements* and *crazytaxi*. The reported values are in bytes and milliseconds and are presented in Tables 5 and 6. It should be noted that a Lognormal distribution is described with scale (σ), location (μ), and threshold (γ) parameters, and Weibull distribution described with scale (λ), shape (k) and right limit. Lognormal, Weibull, Uniform, and deterministic distributions are designated with Logn., Weib., Unif., and Deter., respectively. Each table comprises: (1) the traffic direction; (2) the SSRC of the modelled flow; (3) the acronym of the flow; (4) the size of the dataset (i.e. number of units on which the modelling procedure has been carried out); (5) the model of packet sizes (PS) and IATs and the probabilities of specific distribution if split modelling is applied; and (6) the values of the model parameters.

As observed in Tables 5 and 6, the server's CBR-Audio and Cursor flows are game independent. The majority of flows have constant packet rates (i.e. a fixed IAT). Only the user input (Keys flow) and Video flow show a significant dispersion of the IAT. The traffic flow carrying player's keys depicts two characteristic IAT values, 10 and 50 ms, the latter being an upper limit (i.e. the game does not send the packets of this flow at a lower rate than 20 packets per second). These two values represent more than 50 % of the overall CDF for *4elements*, whereas it is only 30 % of CDF for *crazytaxi*, of which the remaining values are uniformly distributed between 0 and 40 ms. The most significant difference between games is in server-side Video flows. Packet sizes of both games are modelled with a combination of Uniform distribution from 0 to 1,355 bytes and a deterministic portion at 1,356, which is the highest value of packet size appearing in the video flow. Video flow comprises a certain part of subsequent packets with near 0 ms IAT values, while the remaining packets are modelled using a Lognormal distribution. VBR-audio stream consists of two values: a lower endpoint at 244 bytes and higher endpoint at 1,384 bytes. Also, in *4elements* we observe 30 % of the packets uniformly distributed between these two endpoints, while in *crazytaxi* there are almost no such packets.

As for the Video burst sizes, we have observed for both games that they are well approximated with a three parameter Lognormal distribution; as indicated by Fig. 5. It can be assumed that these two CDF lines present limits of Video burst sizes between which all other OnLive games would fit, as these two games represent the two limits (i.e. games with highest and lowest bandwidth usage out of all the inspected games). With regard to Video burst IATs, they are shown to be very regular and can be well approximated with a deterministic value of 16 ms.

As for the client side, the packet sizes of both games are well approximated with a Uniform distribution. *Crazytaxi* has a slightly wider set of values for packet sizes sent from the client. It is interesting to note that the IATs of the Keys

Table 5 Network traffic model parameters for *crazytaxi*

Direction	RTP SSRC	Acronym	Count	PS model	PS parameters (bytes)	IAT model	IAT parameters (ms)
Downstream	0x30000	CBR-Audio	14,710	Deter.	a = 216	Deter.	a = 10
	0x40000	Cursor	3,156	Deter.	a = 28	Deter.	a = 50
	0x50000	VBR-Audio	2,460	Deter. p = 7.76 %	a = 244	Deter.	a = 50
				Deter. p = 92.24 %	a = 1,384		
	0x60000	Video	9,5718	Unif. p = 36.06 %	$a = 0, b = 1,355$	Deter. p = 57.25 %	$a = 0$,
				Deter. p = 63.94 %	$a = 1,356$	3-Logn. p = 42.75 %	$\mu = 0.34$, $\sigma = 1.73$, $\gamma = -2.25$
Upstream	0x0XXXX	Keys	5,107	Unif.	$a = 25, b = 210$	Deter. p = 32.31 %	$a = 50$
						Weib. p = 67.69 %	$\lambda = 22.70$, $k = 1.33$, limit = 50

Table 6 Network traffic model parameters for *4elements*

Direction	RTP SSRC	Acronym	Count	PS Model	PS Parameters (bytes)	IAT Model	IAT parameters (ms)
Downstream	0x30000	CBR-Audio	18,254	Deter.	$a = 216$	Deter.	$a = 10$
	0x40000	Cursor	3,931	Deter.	$a = 4$	Deter.	$a = 50$
	0x50000	VBR-Audio	3,958	Deter. $p = 5.32$ %	$a = 244$	Deter.	$a = 50$
				Unif. $p = 30.28$ %	$a = 245$, $b = 1{,}383$		
				Deter. $p = 64.4$ %	$a = 1{,}384$		
	0x60000	Video	62,714	Unif. $p = 73.93$ %	$a = 0, b = 1,355$	Deter. $p = 23.23$ %	$a = 0$,
				Deter. $p = 26.07$ %	$a = 1,356$	3-Logn. $p = 75.77$ %	$\mu = 0.89$, $\sigma = 122.06$, $\gamma = -3.89$
Upstream	0x0XXXX	Keys	5,107	Unif.	$a = 25, b = 170$	Deter. $p = 43.91$ %	$a = 9$
						Deter. $p = 9.36$ %	$a = 50$
						Weib. $p = 46.73$ %	$\lambda = 12.40$, $k = 0.89$, limit = 50

flow are limited to 50 ms. This means that packets in this flow are sent at a faster rate (i.e. the user's driving input is very fast), but they will never be sent at a smaller rate than 20 packets per second. These differences between flow characteristics on the input side can be attributed to the nature of the game (racing vs. puzzle).

As previously stated, we have modelled specific RTP flows, and examined how accurate the model is by calculating the discrepancy measure at the UDP aggregated traffic level. The computation of the discrepancy for all aggregated traffic is simple for packet sizes, as packet sizes do not change with flow aggregation. On the other hand, IAT values for aggregated traffic are dependent on the interaction between different flows. Due to deterministic modelling, a problem occurs if all flows are started at the same time—extremely bursty traffic is created because of 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 IAT of the aggregated traffic. To alleviate this problem, we need to start generating separate RTP flows at different points in time. We extracted the starting points of separate flows from the recorded empirical datasets in the order of hundreds of milliseconds, and traffic generation procedure was started based on those empirical timestamp values.

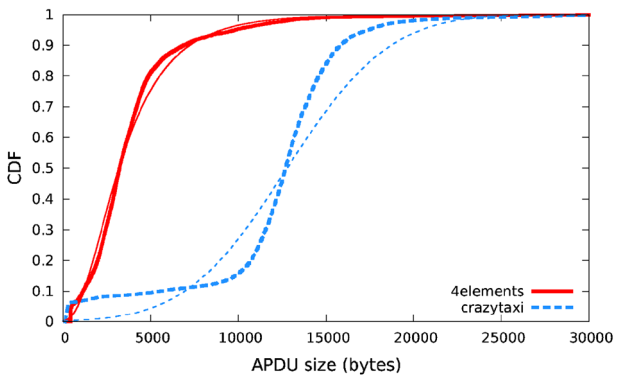


Fig. 5 CDFs of empirical APDU sizes (*thicklines*) and fitted three-parameter Lognormal distribution (*thinlines*) for video flows of *4elements* and *crazytaxi*

In Tables 7 and 8, we present the goodness of fit for 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 the 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 ξ parameter), and whether the model underestimates the tail "-" and "0"(which appears in the case of models that 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 can be ascertained that most of the models have very good fits (i.e. very small discrepancy values). Only the models for the packet sizes of upstream traffic show less accurate fits (i.e. high discrepancy values). This is due to packet sizes being modelled with a uniform distribution resulting in a straight line

Table 7 Goodness of fit for *crazytaxi* traffic models

Direction	Parameter	Discrepancy	Tail	Tail %	ξ	Estimation	ACF
Downstream	Packet size	0.02064	6	0.00433	0	0	0.09376
Downstream	IAT	0.07081	53	0.03823	−1.72591	–	0.03296
Upstream	Packet size	57.53403	253	3.84907	0	0	-0.00630
Upstream	IAT	0.38168	35	0.51546	0	0	0.01755

Table 8 Goodness of fit for *4elements* traffic models

Direction	Parameter	Discrepancy	Tail	Tail %	ξ	Estimation	ACF
Downstream	Packet size	0.12806	6	0.00555	0	0	0.22845
Downstream	IAT	0.05063	51	0.02868	−1.70927	–	0.12358
Upstream	Packet size	117.97917	43	0.29	0	0	0.00912
Upstream	IAT	0.08321	12	0.08129	0	0	0.11913

(model) through stepwise function (empirical data) in the CDFs, as observed in Figs. 6a and 7a. Although the discrepancy values of these two models are high, we think that these models are appropriate because (a) they are sufficiently general to be applied to different games, and (b) since the packet sizes in the upstream are very small, even significant discrepancies in the model will cause only small variances in the upstream bandwidth usage. Furthermore, the amount of data observed in tails is negligible. The values of ξ indicate that the model under/over estimation of the empirical data is not severe. The correlation values presented are very low. Interestingly, we have found higher correlations for the IATs of upstream packets at lag 7 for both games (i.e. correlations values around 0.4). To conclude, it should be pointed out that the mathematical description of the models' goodness of fit illustrates that the models presented are good fits, providing a quality in line with similar models found in the literature (e.g. [13]).

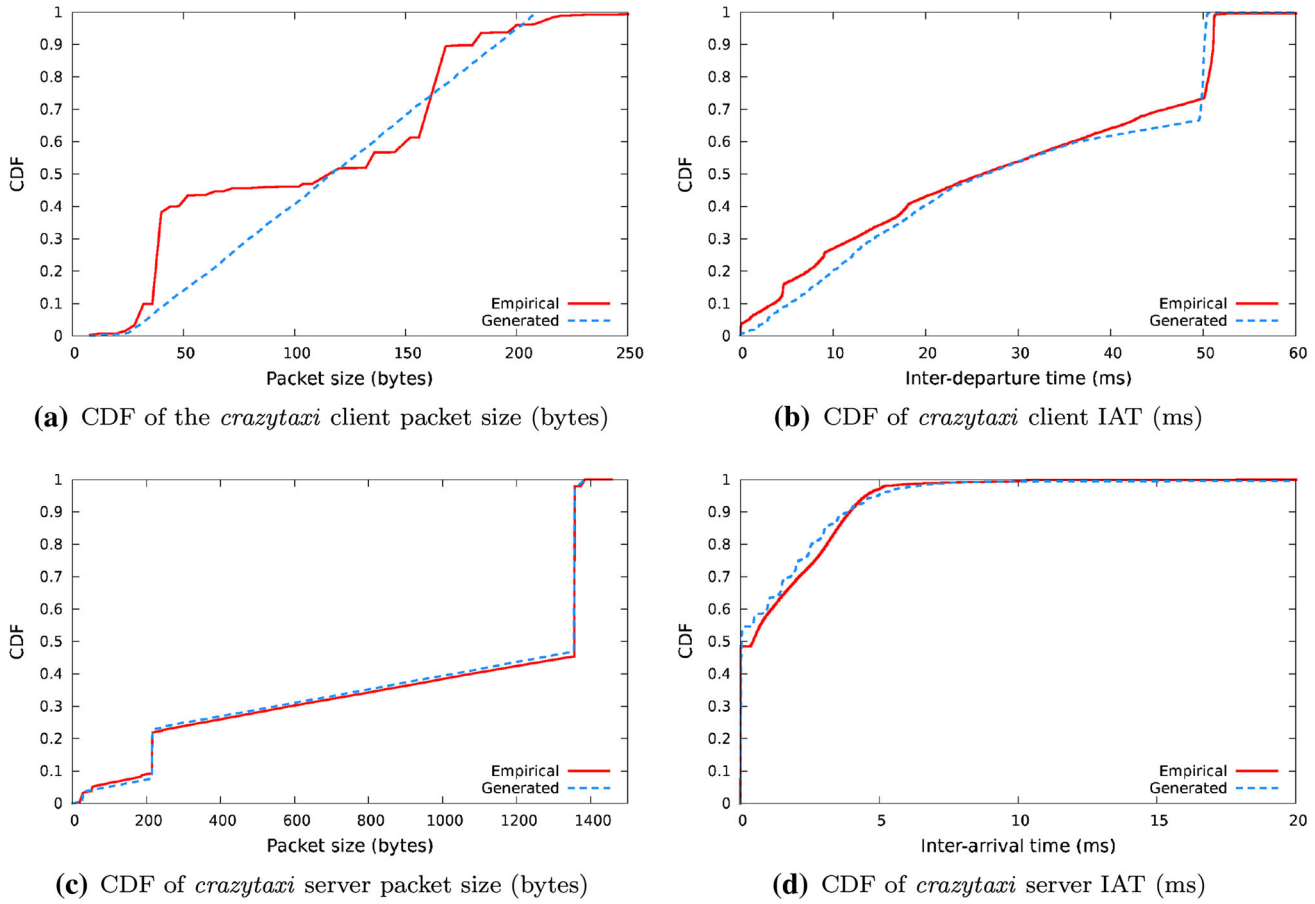
5.3 Experimental validation

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) [41]. 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 behavioural parameters of UrBBaN-Gen to generate one flow of each game for the duration of the whole experiment. For traffic generation in UrBBaN-Gen, we use the Distributed Internet Traffic Generator (D-ITG) developed by the University of Naples [1].

To use UrBBaN-Gen to generate the 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@2.3 GHz 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 Figs. 6 and 7, we present the results of our modelling procedures for *crazytaxi* and *4elements*, respectively. Both figures are composed of four graphs. Each depicts one of the four parameters modelled: client-side packet size, client-side IAT, server-side packet size and server-side IAT. Each of the graphs contains two CDFs: one from the OnLive traces, and another from the data extracted from the generated traffic. In this way, we can verify our model by comparing generated traffic and real captured traffic.

As observed, the generated traffic closely resembles the captured traffic for most of the parameters presented. The roughest estimation is observed in the client packet sizes (Figs. 6a, 7a), which have a stepwise function indicating that packet sizes which are sent as player input are limited to a set of values. We believe that using uniform distribution as an approximation is well suited, as modelling specific packet sizes does not provide any additional information. In addition, slight discrepancies can be observed in the server side IATs (Figs. 6d, 7d), because of the modelling procedures being performed on separate flows which are then joined as one. The deterministic modelling of the IATs of flows CBR-Audio and Cursor causes "steps" on the CDF of the generated traffic. The



(a) CDF of the *crazytaxi* client packet size (bytes)

(b) CDF of *crazytaxi* client IAT (ms)

(c) CDF of *crazytaxi* server packet size (bytes)

(d) CDF of *crazytaxi* server IAT (ms)

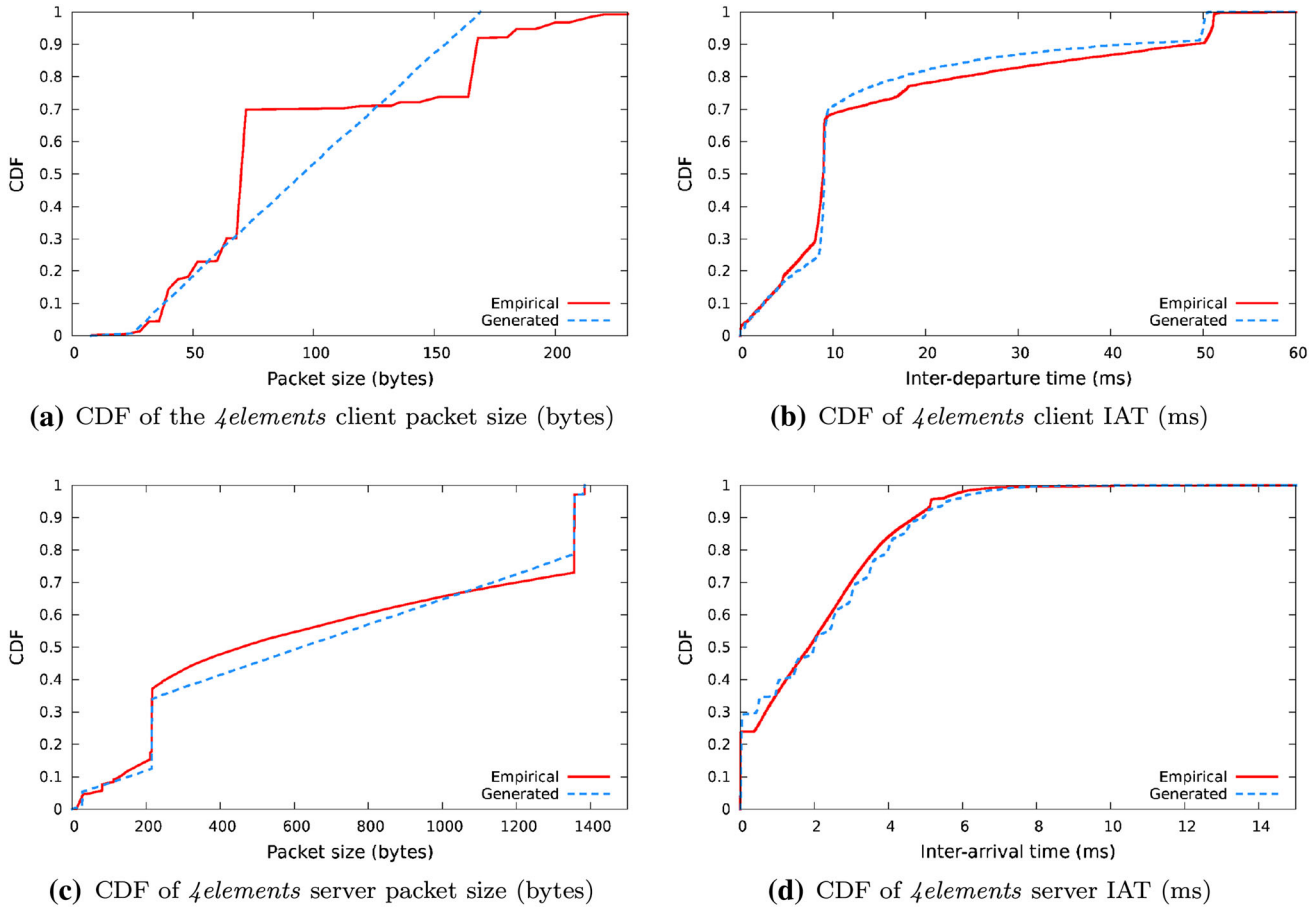
Fig. 6 CDFs of PS and IAT of OnLive traces, the theoretical model and data extracted from the generated traffic

best fits are observed in server-side packet sizes of *crazytaxi* and client side IATs of *4elements* (Figs. 6c, 7b). 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 Figs. 6 and 7.

To summarise, while modelling was performed only on a subset of existing flows, the generated traffic proves to be a good representation of the whole empirical traffic. This confirms that flows which have not been modelled form a negligible portion of the traffic. The models are implemented in the D-ITG network traffic generator, which enables not only tests in a simulated environment, but also in real networks. It is established that Video, VBR-Audio and Keys flows are clearly game dependent. A common model for the two games examined has been provided (i.e. for a specific flow type in each game the same distribution or a combination of distributions is used). Therefore, if more games are modelled, the methodology and distributions presented here can be employed which, in turn, makes the overall modelling task much simpler. For flows common to all games on the OnLive platform, the models provided in this paper can be used as given, whereas for the varying flows provided, the distributions and their combinations can be used, although their parameters may need to be adjusted.

6 Summary and future work

Cloud gaming, as a new approach to the world of online games, heavily modifies the properties of network traffic by increasing both the requirements of bandwidth used by a factor of 100 or more, as well as of network latency. A comparison of different cloud gaming platforms is needed to identify any general properties of cloud gaming traffic. However, comparing two different cloud gaming platforms first requires the understanding of the employed protocols, and since most protocols used by the cloud gaming providers are proprietary, the only way to understand them is by reverse engineering. In this paper, we dissect the protocol of one of the main cloud gaming platforms—OnLive. We have collected and analysed extensive traffic traces from several video games of different genres. It has been



(a) CDF of the *4elements* client packet size (bytes)

(b) CDF of *4elements* client IAT (ms)

(c) CDF of *4elements* server packet size (bytes)

(d) CDF of *4elements* server IAT (ms)

Fig. 7 CDFs of PS and IAT of OnLive traces, the theoretical model and data extracted from the generated traffic

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. We have fully characterised the traffic of the last phase (being the most important one in terms of time players spend in it and of generated traffic load). A detailed analysis is performed on two video games (*crazytaxi* and *4elements*) that have the most dissimilar traffic profiles. We have identified the following RTP flows in the OnLive traffic: Monitor, Control, CBR-Audio, Cursor, VBR-Audio, Video, and Chat in the downstream direction, and Keys, Mouse Control-ACK, and Mic in the upstream direction. The Video flow generates the largest network traffic load. 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 accurately describes the OnLive traffic.

For future work, we plan to conduct a reverse engineering of the Gaikai protocol which will enable us to compare two different cloud gaming platforms and extract the common characteristics of cloud gaming traffic between the two platforms. Additionally, we plan to investigate different network scenarios (e.g. wired/mobile, bandwidth, delay, jitter, etc.) and how they impact such cloud gaming traffic characteristics.

Acknowledgments This work was 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, by the Ministry of Science, Education, and Sports of the Republic of Croatia, and Ericsson Nikola Tesla, Zagreb, Croatia. The research leading to these results has also 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. Comput. Commun. **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. Commun. Mag. **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.: Real-time 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. Huang, C.Y., Hsu, C.H., Chang, Y.C., Chen, K.T.: Gaminganywhere: an open cloud gaming system. In: Proceedings of ACM SIGMM Conference on Multimedia Systems (MMSys'13) (2013)
9. Cisco Systems Inc: Cisco Visual Networking Index: Forecast and Methodology, pp. 2011–2016 (White paper) (2012)
10. Claypool, M., Claypool, K.T.: Latency and player actions in online games. Commun. ACM **49**(11), 40–45 (2006)
11. 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)
12. Cricenti, A., Branch, P.: ARMA(1,1) modeling of Quake4 server to client game traffic. In: Proceedings of the 6th ACM SIGCOMM Workshop on Network and System Support for Games, pp. 70–74 (2007)
13. Dainotti, A., Pescapé, A., Ventre, G.: A packet-level traffic model of Starcraft. In: Proceedings of the Second International Workshop on Hot Topics in Peer-to-Peer Systems, pp. 33–42 (2005)
14. DFC Intelligence. http://www.dfcint.com/wp/?p=277. Accessed Online 27 March 2013
15. Färber, J.: Network game traffic modelling. In: Proceedings of the 1st Workshop on Network and System Support for Games, pp. 53–57 (2002)
16. Feng, W.C., Chang, F., Feng, W.C., Walpole, J.: A traffic characterization of popular on-line games. IEEE/ACM Trans. Netw. **13**(3), 488–500 (2005)
17. 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)
18. Hedayat, K. et al.: A two-way active measurement protocol (TWAMP) (2008) (updated by RFC 5618)
19. 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 Trans. Graph. **21**, 693–702 (2002)
20. Karachristos, T., Apostolatos, D., Metafas, D.: A real-time streaming games-on-demand system. In: Proceedings of the 3rd International Conference on Digital Interactive Media in Entertainment and Arts, pp. 51–56 (2008)
21. 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)
22. 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, pp. 102–107 (2011)
23. KwangSik, S., et al.: Transformation approach to model online gaming traffic. ETRI J. **33**(2), 219–229 (2011)
24. Lakkakorpi, J., Heiner, A., Ruutuc, J.: Measurement and characterization of Internet gaming traffic. In: Technical Report, Espoo, Finland. Research Seminar on Networking (2002)
25. 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)
26. Lang, T., Armitage, G., Branch, P., Choo, H.Y.: A synthetic traffic model for half-life. In: Proceedings of the Australian Telecommunications, Networks and Applications Conference (2003)
27. Lang, T., Branch, P., Armitage, G.: A synthetic traffic model for Quake3. In: Proceedings of the 2004 ACM SIGCHI International Conference on Advances in computer entertainment technology, pp. 233–238 (2004)
28. 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)
29. 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)
30. Odlyzko, A.M.: Internet traffic growth: sources and implications. In: Proceedings of SPIE, pp. 1–15 (2003)
31. OnLive. http://www.onlive.com. Accessed Online 27 February 2013
32. Park, H., Kim, T., Kim, S.: Network traffic analysis and modeling for games. Internet and Network Economics. Lecture Notes in Computer Science, pp. 1056–1065. Springer, Berlin (2005)
33. Paxson, V.: Empirically-derived analytic models of wide-area TCP connections. IEEE/ACM Trans. Netw. **2**(2), 316–336 (1994)
34. Pederson, S.P., Johnson, M.E.: Estimating model discrepancy. Technometrics **32**(3), 305–314 (1990)
35. 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)
36. Ratti, S., Hariri, B., Shirmohammadi, S.: A survey of first-person shooter gaming traffic on the internet. IEEE Internet Comput. **14**(5), 60–69 (2010)
37. Sandvine: Global internet phenomena report. In: Technical Report (2013)
38. Shalunov, S. et al.: A one-way active measurement protocol (OWAMP) (2006)
39. Shea, R., Liu, J., Ngai, E.C.H., Cui, Y.: Cloud gaming: architecture and performance. IEEE Netw. **27**(4), 16–21 (2013)
40. Suznjevic, M., Dobrijevic, O., Matijasevic, M.: MMORPG player actions: network performance, session patterns and latency requirements analysis. Multimed. Tools Appl. **45**(1–3), 191–241 (2009)
41. Suznjevic, M., Stupar, I., Matijasevic, M.: A model and software architecture for MMORPG traffic generation based on player behavior. Multimed. Syst. (2012)
42. Svoboda, P., Karner, W., Rupp, M.: Traffic analysis and modeling for world of warcraft. In: Proceedings of the IEEE International Conference on Communications, pp. 1612–1617 (2007)
43. The Telegraph: http://www.telegraph.co.uk/technology/. Accessed Online 27 March 2013
44. 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)

45. 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)
46. 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)
47. Yoo, C.S.: Cloud computing: architectural and policy implications. Rev. Ind. Organ. **38**(4), 405–421 (2011)