Reducing the Impact of Massive Multiplayer Online Games on the Internet using SCTP

David Burgos-Amador, Jesús Martínez-Cruz, Sergio Recio-Pérez
Dpto. Lenguajes y Ciencias de la Computación
University of Málaga, Spain

***Abstract*—Games are on the Internet to stay. Their popularity and use are growing every day, and some genres such as the Massive Multiplayer Online Role-Playing Games (MMORPGs) have become an important source of network traffic having several thousands of simultaneous players at any one time. These kinds of games constitute a real challenge for developers but also for Internet service providers due to their high requirements of bandwidth and hardware resources. Therefore, it is really important for MMORPGs to be supported by high-performance network code. We show how these games can benefit from the new transport protocol in the TCP/IP stack, the Stream Control Transmission Protocol (SCTP) which can increase the overall efficiency of a game protocol in terms of latency and jitter, along with extra savings in processing time for packet management. Both of these are important details for player's game experience and for server resource provisioning, respectively.**

***Keywords*-protocols, network traffic, multiplayer games, performance**

I. Introduction

Massive Multiplayer Online (MMO) games are now one of the most important sources of Internet traffic which is rapidly growing every year. By 2008, MMO games represented 30% of the overall network traffic in the US backbone [1]. Some genres such as the MMO role-playing games (MMORPGs) constitute a real challenge for networks and service platforms. Undoubtedly, the most important example of this trend is Blizzard's World of Warcraft [2], which holds the Guinness record for the number of subscribers, which currently stands at more than 11.5 millions. Therefore, users demand a smooth game experience, with network latencies of less than 1.5 seconds, which becomes a really stringent constraint for our old best-effort IP networks. Moreover, MMORPG servers must be executed on expensive clusters to support hundreds of thousands of simultaneous players. Obviously, the number of clusters and bandwidth must be continually growing in order to satisfy the demand of new subscribers. Nevertheless, the question arises as to whether there is an alternative solution which can optimize the games resources without imposing any extra cost on network infrastructures and game platforms. This article shows how the application of new features available in the SCTP transport protocol allows MMORPGs to perform better in terms of network latency but also in terms of CPU and hardware resources savings. Therefore, we describe our experience with porting an open source MMORPG called Planeshift [3] to SCTP (see fig. 1). Our results are also applicable to other application domains where users demand low-latency responses within environments with a high probability of packet losses.

Figure 1. A Planeshift screenshot

II. MMORPGs meet TCP/IP protocols

MMORPG are Internet applications which follow the Client/Server model. They must exchange data through the available transport protocols of the TCP/IP stack, mainly TCP and UDP. However, the differences between these two protocols and the particular features of MMORPGs make their selection and use a non-trivial task.

UDP is a connectionless protocol that offers an unreliable delivery service for application packets, which may arrive unordered, duplicated, or be lost without further notice, making applications responsible for providing such guarantees, whenever needed. However, its main advantages are the low overhead it involves, the possibility of communicating with any number of nodes through the same end-point (using only one socket) and its support for broadcast and multicast delivery. In contrast, TCP is a connection-oriented protocol that provides a reliable byte-stream delivery service and

978-0-7695-4967-5/13 $26.00 © 2013 IEEE
DOI 10.1109/ITNG.2013.9

congestion control. However, it incurs a higher overhead and usually increments the associated jitter (variations of the packet latency) due to the problem known as head-of-line blocking, which is caused by its ordered delivery.

Therefore, the UDP protocol has been commonly used in fast-paced first person action games because of their need for short response times and their tolerance to packet loss. TCP, in turn, is widely used in strategy games, where there is not such a strict requirement in response times but it is important to follow an ordered sequence of specific events, because packet losses affect the player's game experience [4]. But what about MMORPGs? These combine features from both action and strategy games, all mixed in a virtual world full of social interactions. The traffic patterns that these games produce is very different from the traffic of dominant Internet applications such as file transfer or web browsing, and also from other types of Internet multiplayer games. Their protocols usually exchange small packets, where each type of data packet is classified based on its importance and priority within the game (including those which are sent periodically). Compared to other kinds of games, they also require a lower average bandwidth for clients although a positive correlation exists between the generated network traffic and the number of active server connections (that is, the number of simultaneous players). In this context, the overhead introduced by a connection-oriented protocol such as TCP becomes prohibitive for some MMORPGs (e.g. in ShenZhou Online, the overhead of TCP headers represent 73% of the total transmitted bytes and the *ack* segments (acknowledgement packets in TCP) 30% of the total [5]). However, any alternative based on UDP must deal with its unreliable nature and, therefore, the application layer game protocol has to implement some reliability mechanisms by itself. This is why currently there is no consensus among developers on which transport protocol is the best choice for MMORPGs.

That being that the case, lots of MMORPGs use UDP, such as EverQuest, Star Wars Galaxies, City of Heroes, Ultima Online, Asheron's Call, Final Fantasy XI or Planeshift, but also a similar number of them use TCP, such as World of Warcraft, Lineage I and II, Guild Wars, Ragnarok Online, Anarchy Online and Mabinogi. There are even some rare cases, such as Dark Age of Camelot, which use a combination of both TCP and UDP.

III. Sockets for SCTP

The Stream Control Transmission Protocol is now a general-purpose transport protocol which is available in many platforms. SCTP shares some similarities with TCP and UDP, but also supports new features such as multi-streaming, multihoming or partially ordered delivery among others, it being now possible to develop SCTP-aware network applications using the common socket API along with some ad-hoc extensions [6]. We can define two types of SCTP sockets: the *one-to-one*, which follows the TCP operation model and represents an association between two peers, and the *one-to-many*, which is similar to UDP and can handle multiple simultaneous associations on the same socket. The *one-to-many* socket type provides full support for a *sctp_peeloff* feature: an association can be peeled off from a one-to-many socket to its own one-to-one socket which can be managed in a specific execution thread or forked process. Both types of sockets support a notification system which reports about transport-level events, including network status changes, remote operational errors, association startups or undeliverable messages. It is possible to set which events are to be subscribed using an *sctp_event_subscribe* data structure and passing its values to the `setsockopt()` socket function. This can help an application to better manage the associations it maintains. When reading data from the socket, the flags must be checked to verify if the received data is a notification or a message. In the case of a notification, an *sctp_notification* data structure is returned to indicate its type.

SCTP sockets can be also configured to control low-level connection details such as the maximum transfer unit, heartbeat interval or selective *ack* delay, using an *sctp_paddrparams* data structure with the `setsockopt()` function. Developers can also control the way in which streams will send and receive data, using an *sctp_sndrcvinfo* data structure with the special `sctp_sendmsg()` and `sctp_recvmsg()` functions.

IV. SCTP in MMROPGs: the Planeshift case

Planeshift is an open source MMORPG, which has been developed in C++ and is available for many platforms, such as Unix/Linux, Mac OS and Windows. This game is free of charge, and has two official servers active with more than four hundred thousands of subscribers. The availability of its source code makes it possible to redesign it to include support for SCTP, then analyzing changes in its behavior and performance. The software architecture of Planeshift contains a class called NetBase, which manages the communication procedures code for both client and server. For each game, a single NetBase object instance controls the sending and reception of data in its own thread of execution. The understanding of the NetBase class behavior and the game protocol in Planeshift has provided us with all necessary details to design and implement our porting to SCTP.

A. Communications in the original Planeshift code

The Planeshift game protocol relies on UDP arguing that it performs better than TCP. However, as we discussed previously, MMORPGs must include some mechanisms to guarantee the reliability of certain kinds of packets. Therefore, Planeshift implements its own procedures to acknowledge received data. The game information exchanged between client and server (in XML format) is classified

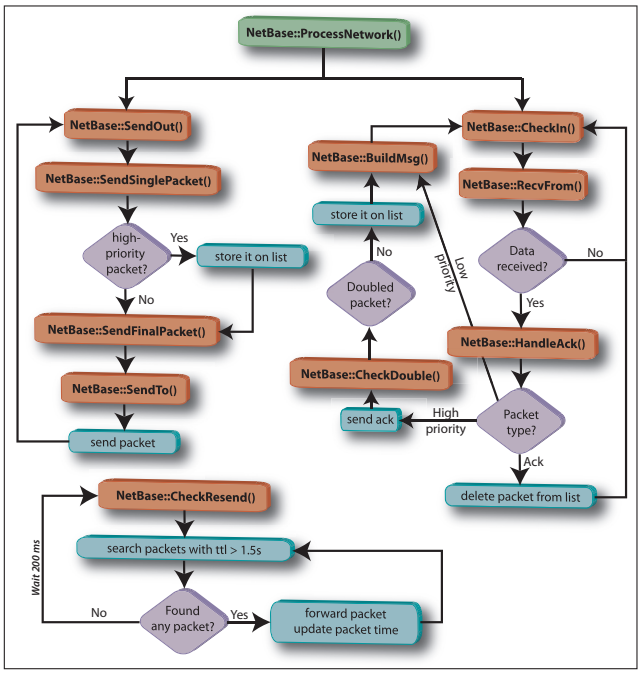

Figure 2. Communications in Planeshift: flow diagram of the NetBase class

Setting initial configuration

```
struct sctp_event_subscribe evnts;
evnts.sctp_data_io_event = 1;
evnts.sctp_association_event = 1;
setsockopt(mysocket, IPPROTO_SCTP, SCTP_EVENTS, &evnts, sizeof (evnts));
struct sctp_paddrparams sp;
sp.spp_hbinterval = 3000;
setsockopt(mysocket, IPPROTO_SCTP, SCTP_PEER_ADDR_PARAMS, &sp, sizeof(sp));
listen(mysocket, LISTENQ);
```

Send data

```
struct sctp_sndrcvinfo sri;
if (pkt->packet->GetPriority() == PRIORITY_HIGH) {
       sri.sinfo_stream = 0;
       sri.sinfo_flags = SCTP_UNORDERED;
} else {
       sri.sinfo_stream = 1;
       sri.sinfo_flags = SCTP_PR_SCTP_TTL | SCTP_UNORDERED;}
sentbytes = sctp_sendmsg (mysocket, data, size, addr, tolen, 0 , sri->sinfo_
flags, sri->sinfo_stream, 1500, 0);
```

Receive data

```
int err = sctp_recvmsg(mysocket, buf,MAXPACKETSIZE, (sockaddr *) recaddr, /
len, sri, msg_flags) ;
if (msg_flags) {
   union sctp_notification *snp; struct sctp_assoc_change *sac;
   snp = (union sctp_notification *) buf;
   switch (snp->sn_header.sn_type) {
       case SCTP_ASSOC_CHANGE:
          sac = &snp->sn_assoc_change;
          switch (sac->sac_state) {
              case SCTP_COMM_LOST: CheckLinkDead();
              break;}}}
```

Figure 3. Porting Planeshift to SCTP: source code excerpts

and encapsulated into three types of packets: *i)* high-priority packets, which carry game data and must be acknowledged after reception (or retransmitted in the case of loss); *ii)* low-priority packets, which also carry game data but do not require either acknowledgement or retransmission; and finally *iii) ack* packets, which are used to notify peers on the arrival of high-priority packets. Depending on the action that takes place in the virtual world, the application level will map it into one type of packet or another. The important actions of the game such as player movements or commerce transactions will be sent in high-priority packets, while periodic messages associated with the world state (such as objects positions, weather, etc.) will be sent in low-priority packets. The game protocol header in Planeshift is composed of a packet identifier (a number that distinguishes it from the rest), a payload type identifier and a priority bit.

Regarding the game protocol behavior implemented in the NetBase module, fig. 2 shows its flow diagram. The starting point is the `ProcessNetwork()` method, which is responsible for managing both sending (through the `SendOut()` method) and receiving data (using the `CheckIn()` method).

When some game information is ready to send, `SendOut()` creates the packet and first proceeds by calling `SendSinglePacket()`, which adds the packet to a list of confirmation-pending ones if it is of high-priority type. Next, `SendFinalPacket()` writes the packet on the socket for its delivery (assisted by the `SendTo()` helper method). In order to retransmit packets that have reached their time-to-live timeout (established in 1.5 seconds), `CheckResend()` is called every 200 milliseconds.

The `CheckIn()` procedure first checks on the reception of packets from the socket, after which it executes the `HandleAck()` method. This function verifies if the received packet is an *ack* (then removing a corresponding packet from the pending list). Otherwise, the `CheckDoublePackets()` method will discard the packet if it recognizes it as a duplicate (using its packet identifier as a matching filter). The payload of valid packets will be passed to the game in order to parse and process its contents. It is worth noting that `CheckIn()` is also responsible for sending an *ack* packet after the reception of a high-priority one, as depicted in fig. 2. Besides the functionality described, NetBase also sends periodic packets every 3 seconds to verify the availability of the peer and the link status.

B. Adapting Planeshift to SCTP

In order to get an SCTP-aware version of Planeshift we have chosen the (Unix-like) Mac OS X Snow Leopard platform and a typical *gcc*-based toolchain. SCTP is available for this operating system as a network kernel extension [7] developed by Michael Tuexen.

Therefore, our first changes in the code have been focused on the creation of the SCTP socket, which can reuse and adapt the existing sequence of socket operations (socket, bind, listen, ...) with the appropriate arguments for SCTP. Then, we need to establish the configuration of the socket

using `setsockopt()`. In this configuration we will activate the I/O and association-related notifications through the desired events available in the *sctp_event_subscribe* structure (see fig. 3). The code shows the activation of the SCTP socket for sending and receiving data along with the notification of any association changes in clients. The *sctp_paddrparams* structure is also used to set the heartbeat interval to 3 seconds, in order to check the link status between server and client as it was made in the original Planeshift game protocol. As we have discussed previously, the use of UDP in Planeshift requires that part of the game protocol management regarding reliability must be done at the application level. However, using the SCTP transport protocol will make the functionality in `HandleAck()`, `CheckDoublePackets()` and `CheckResend()` no longer necessary, since the protocol itself provides it.

For packet sending, `SendOut()` will have only to pass the packet to the socket using the *sctp_sendmsg* function, now with two extra arguments: the time-to-live value of the packet (TTL) and the stream selected for delivery. As seen before, the time-to-live value for packets in Planeshift was 1.5 seconds, so we will specify it here. Regarding data streams, we have decided to create two of them: a first one to deliver high-priority packets, which will be configured as reliable and not ordered, and a second one for low-priority packets, which will be configured as TTL partially reliable and not ordered. It is worth noting that every stream can be configured in an independent and flexible way, which fits well with the requirements of Planeshift and other MMORPGs, as also mentioned in [8]. As packets are delivered unordered, we also avoid any head-of-line blocking issues. Fig. 3 depicts this implementation using a *sctp_sndrcvinfo* structure, where the *sinfo_stream* field sets the stream identifier to use, and the *sinfo_flags* field indicates how the stream is configured.

In order to receive data, the `CheckIn()` method must read from the socket using the *sctp_recvmsg* function and check if the received content is a notification or a game packet. Any change in an SCTP active association is notified using *msg_flags*, as depicted in fig. 3, where the code included shows how to discover the right type of notification. When the association does not respond to heartbeats, it is interpreted that it has become inactive and then the `CheckLinkDead()` method will manage this failure.

V. Experiments and Results

MMORPGs have a severe impact on Internet service providers, but also in the Server hardware infrastructure required to support thousands of concurrent players. After our changes to the Planeshift source code we wanted to check the result for improvements in two key areas: *i)* network performance and *ii)* CPU consumption of network code on the server side.

We have used the Dummynet tool [9] in our experiments, which can simulate a network cloud with specific values of average latency, bandwidth and percentage of packet loss. We have also used the Wireshark protocol analyzer [10] on the server side to record and analyze network traffic.

In our first test we run a series of games with one client and one server, using both the UDP and SCTP versions in two Apple's MacBook computers with an Intel Core 2 Duo processor. With the aid of Dummynet, we established a bandwidth of 10 Mbps, an average latency of 300 ms, and a percentage packet loss of 0.1%, 1%, 2% and 5%. After the experiments, we used the resulting Wireshark trace files to extract information about the performance of the UDP and SCTP games (with two specific *awk* scripts). Therefore, we measured the latency experienced by packets in terms of the time elapsed since the packet is first sent until it is acknowledged. It is worth noting that (selective) *ack* packets in SCTP can acknowledge more than one packet at the same time (within a predefined sack delay interval), which influences the measured latency value of the packets that fall into this interval, except for the last one. We also recorded the total size of bytes sent and received at transport and application level (throughput and goodput, respectively) in order to measure and contrast the efficiency of both transport protocols (defined as the goodput divided by throughput). Lost and retransmitted packets were also taken into account for the elaboration of the statistics.

Fig. 4 shows the results obtained in the first test scenario. The four graphics depict the latency figures with different percentages of packet loss. As we can see, regardless of losses, SCTP always obtained a lower latency that remained generally below 1.5 seconds. Packets which did not suffer any loss obtained a value similar to the network latency (300 ms). The figures show how SCTP packets seem to have reached a higher latency value between the interval from 0.3 to 0.5 seconds in the y-axis (the red area depicted), which was due to the sack delay interval. Upon a packet loss, latency in SCTP was usually close to 1 second but always lower than 1.5 seconds (the subjective limit for a satisfactory game experience). On the other hand, UDP delays were close to 2 seconds and sometimes they doubled this value. In fact, the average latency in the scenario with a loss level of 5% was 1.459 seconds when using SCTP, and 2.030 seconds when using UDP, which is 40% more.

When considering the protocol efficiency, SCTP obtained 14-20% more than UDP regardless of the loss level. The explanation for this result is the capacity of SCTP to group different game packets within a bigger transport packet when they all have to be sent within a short period of time. This resulting transport packet then shares a common header for all the contained packets (20 bytes), where each one has its own specific header (8 bytes). The same is valid for *ack* packets, which can acknowledge more than one packet but can also transport data. In UDP instead, each game packet

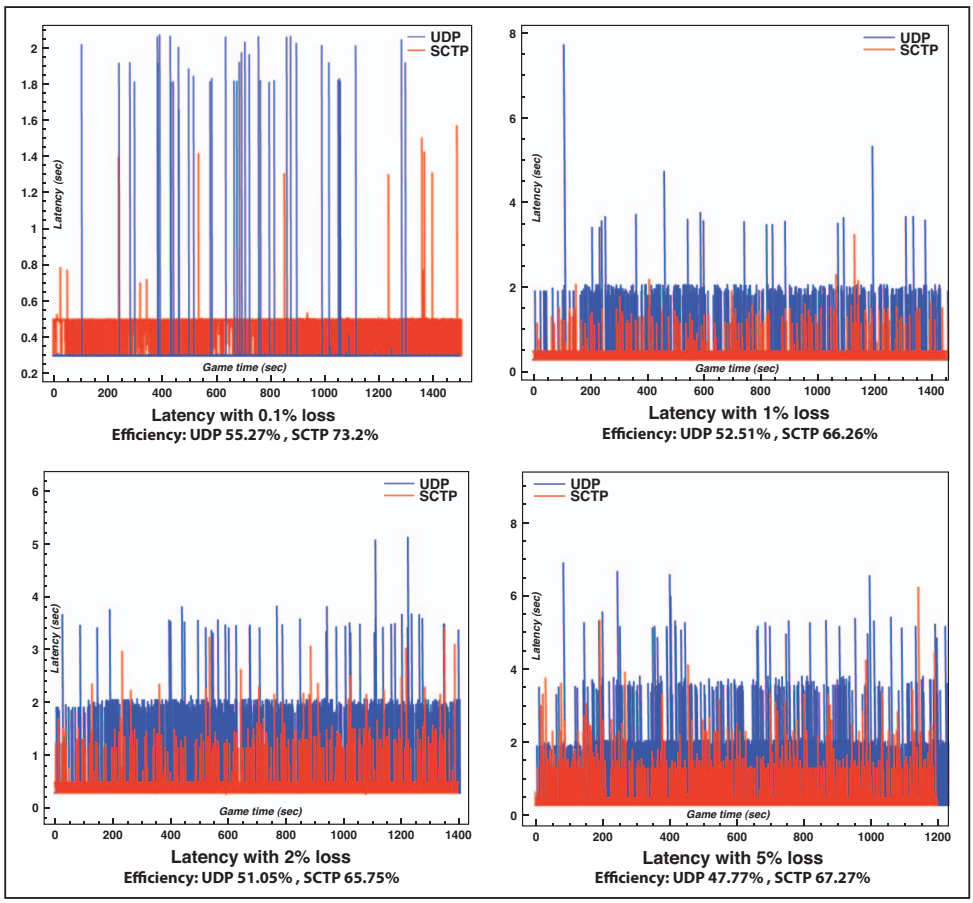

Figure 4. Latency and performance results comparing SCTP and UDP with one client

is encapsulated in a datagram that has a header of 8 bytes. The *ack* game packets can only acknowledge one packet at a time. This causes the average packet size to be 624 bytes in SCTP and 96 bytes in UDP, allowing SCTP to be more efficient despite having a larger header.

After these promising results, we moved into a second test scenario now with ten simultaneous players. Fig. 5 shows our results in a network context with 10 Mbps of bandwidth, 300 ms of average latency and a loss level of 1%. The number of players was increased in order to observe how they would affect the statistics but also to determine their effect on the server regarding the time spent on its NetBase main methods. The latter was measured using the profiling mechanisms available within the Apple Developer Tools.

The latency in is this scenario was similar to the one obtained in the previous experiment (fig. 5, right). Once more, SCTP obtained a lower latency than UDP, with an average value of 0.884 seconds (for the packets that suffered loss), which is more than two times less than the 2.012 seconds obtained with UDP. We have observed how in this experiment with more players UDP improved its network efficiency to 60.84% mainly due to the increase of low priority packets (2.4% in the first experiment and 30% in this one), but was still beaten by SCTP, which was 14.5% more efficient. The average packet size was 544 bytes for SCTP and 90 bytes for UDP.

As we can see in fig. 5 (left), the `CheckIn()` method consumed 16% more time in UDP than in SCTP. This result was expected because part of the game protocol logic in SCTP is now processed by the kernel, which decreases the overall time spent in managing incoming packets. This extra time has been used for other critical game tasks such as updating the world state and sending data to more players, which implies that the SCTP-aware Planeshift server is now able to handle more simultaneous players than the original one without requiring additional hardware resources.

VI. Conclusions and Future Work

In this example, porting an MMORPG to SCTP has been an easy task. Other developers should also be able to reuse and benefit from a lot of the existing game network code. As shown with Planeshift, the new transport protocol includes new options which adapt well to these games, such as time-to-live, (un)ordered delivery, multistreaming

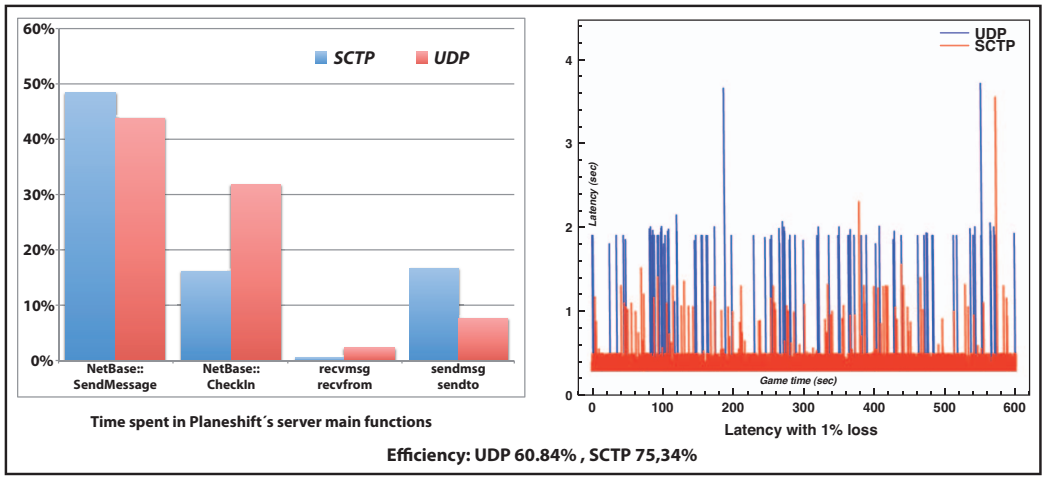

Figure 5. Latency and performance results comparing SCTP and UDP in multiplayer experiments

and multihoming, among others. Most of the game protocol logic will be now done in the kernel, which reduces response times and allows servers to perform better with the same hardware resources. Moreover, our experiments show how latency in SCTP is less than 1.5 seconds, the maximum to allow a satisfactory experience for a game player. This result holds even in network scenarios with non-negligible packet losses. Future work will focus on using and adapting our proposed design methodology to TCP-based MMORPGs and other game genres, such as first-person shooters.

ACKNOWLEDGEMENTS

This work has been partially granted by the Spanish Ministerio de Ciencia e Innovación (MICINN) under project no. TIN2008-05932.

REFERENCES

[1] S. Shirmohammadi and M. Claypool, "Guest editorial for special issue on massively multiplayer online gaming systems and applications," *Multimedia Tools and Applications*, vol. 45, no. 1-3, pp. 1–5, 2009.

[2] Blizzard, "World of Warcraft. Available at http://www.worldofwarcraft.com," 2013.

[3] Atomic Blue, "PlaneShift: the Role Playing Game. Available at http://www.planeshift.it," 2013.

[4] G. Armitage, M. Claypool, and P. Branch, *Networking and Online Games : Understanding and Engineering Multiplayer Internet Games*. John Wiley & Sons, 2006.

[5] K. T. Chen, P. Huang, C. Y. Huang, and C. L. Lei, "Game traffic analysis: an MMORPG perspective," in *NOSSDAV '05: Proceedings of the international workshop on Network and operating systems support for digital audio and video*. New York, NY, USA: ACM, 2005, pp. 19–24.

[6] P. Natarajan, F. Baker, P. D. Amer, and J. T. Leighton, "Sctp: What, why, and how," *IEEE Internet Computing*, vol. 13, no. 5, pp. 81–85, 2009.

[7] Michael Tuexen, "SCTP Network Kernel Extension (NKE). Available at http://sctp.fh-muenster.de/sctp-nke.html," 2013.

[8] C.-C. Wu, K.-T. Chen, C.-M. Chen, P. Huang, and C.-L. Lei, "On the challenge and design of transport protocols for mmorpgs," *Multimedia Tools Appl.*, vol. 45, no. 1-3, pp. 7–32, 2009.

[9] Luigi Rizzo, "Dummynet. Available at http://info.iet.unipi.it/ luigi/dummynet," 2013.

[10] CACE Technologies, "Wireshark network protocol analyzer. Available at http://www.wireshark.org/," 2013.