Design and Simulation of a Migration-based Architecture for Massively Populated Internet Games

Ludovico Gardenghi, Sandro Pifferi, Gabriele D'Angelo, Luciano Bononi
Department of Computer Science, University of Bologna
Mura Anteo Zamboni, 7, 40127 Bologna, ITALY
Email: {gardengl, spifferi, gdangelo, bononi}@cs.unibo.it

Abstract— **In recent years many popular interactive computer games have gained online remote multiplayer functionalities, supported by standard Internet communication protocols and architectures. Due to the heterogeneous communication infrastructures and network asymmetries, some users (i.e. clients) may be suffering slow, congested and unreliable Internet connections, while others may have access to fast and reliable links. A different rate and latency in the delivery of users' commands and event notifications may lead to unfairness issues during the game play, specifically, for the class of real time and interactive games. A dynamic adaptation of the gaming architecture to the limitations of the communication infrastructure could be exploited to reduce these problems. In this paper we present a simple client migration algorithm which can be adopted on a generic multiplayer, multi-server online gaming architecture. Client migration among the servers of the gaming infrastructure is exploited to adapt to the dynamic performances of the general communication network infrastructure. The proposed mechanism has been modeled and simulated for the class of distributed multiplayer and multi-server interactive games, implemented over a general communication network infrastructure. Results show a significant fairness improvement, more homogeneous performances, and the absence of significant overheads.**

I. Introduction

The explosive growth of the Internet, and the even widely deployed access and mobile connectivity to the Internet infrastructure and services, motivated the designers of computer-games to create new, interactive, and distributed multiplayer games, or even to add online multiplayer capabilities to existing ones [1]. The aim of an interactive multiplayer game is to allow a set of players' avatars to interact in a virtual environment, by following specific interaction schemes and game rules. Avatars are implemented as client applications coordinated and managed by a set of distributed servers. The gaming infrastructure is realized by the set of servers, the distributed client applications, the interaction rules and game management protocols, and the interconnecting communication network infrastructure. In simple interactive gaming infrastructures, the interaction policy may be straight-forward: e.g in turn-based games, every client gets its own turn to "move" and every other client would see the action as the expected interaction event (like in a multiplayer cards game). This gaming infrastructure does not have critical network and fairness requirements [2], because every user waits its turn until other users performed their own actions in a predefined order. The more challenging and attractive set of today's multiplayer games lets the users to live and act by proactively generating local and personal actions (i.e. events), which may affect every other user within the game context [3], [4]. The density and the timing constraints of interactions among the dynamic set of clients, obviously depends on the game type and characteristics. It is worth noting that massive, highly interactive games may be critical to be supported in a distributed way, under the communication viewpoint. In this paper we are interested in the class of highly interactive distributed multiplayer games, where each user may take any decision at anytime (i.e. with no predefined turns), like in a virtual battlefield game. The main problem faced in this paper is the heterogeneity and dynamicity in the latency that could exist on the Internet connections among many peer-players connected to the distributed multi-servers architecture supporting the game execution.

A. Work Motivation

Many multiplayer real-time games (e.g. Ultima Online [5]) require that the player sends "commands" (e.g. walk, take objects, fight, shoot) to a virtual character (i.e. the avatar). Commands are messages sent to one server, over the communication infrastructure (i.e. the network), in order to be evaluated and executed. Event execution generates causally dependent actions and environment changes, to be immediately notified to other characters in the gaming environment. Failing to send a command or an event message, or sending it too late, may cause damages to the avatar evolution (death, injury, loss of resources), and may result in unjustified penalties for the player. For this reason, users with high-latency or low-bandwidth Internet connections may be more disadvantaged than their opponents supported by fast connections. The unfairness problem frequently leads to a generalized users' annoyance: slow users get frustrated because they lose easily despite their skills [6], [7], while fast (and fair) users get bored because they win easily thanks to their network performances. This would translate in less users buying the game and subscribing the online gaming services.

The communication protocols adopted for these games usually do not offer specific solutions for this problem. On the other hand, a general conservative approach is to send less

0-7803-8798-8/04/$20.00 ©2004 IEEE

data to slow users [8], thus lowering the game detail. The solution to achieve fairness by reducing the game detail could be unpractical and questionable, under the player satisfaction viewpoint. Other gaming infrastructures could obtain an improved fairness by exploiting network services implemented below the application, without having to touch the internal details of the game protocol. On the other hand, this would require a complete management of the Quality of Service over end-to-end connections, which is not currently implemented and supported on the Internet.

Our proposal in this paper follows a complementary approach. We assume a common network scenario, an online (Internet-based) multiplayer real-time game supported by a distributed multi-server gaming architecture. We argue that the fairness may be improved by acting on the dynamic migration of clients among servers, with minor changes to the game code, and without requiring to rewrite and redesign any communication protocol. Moreover, we do not assume any specific gaming protocol semantics, because we are interested in proposing and evaluating a mechanism that could be adapted to any generalized gaming architecture [9], [10]. In addition to the previous considerations, another motivation for the proposed migration-based approach is explained in the following. Let us assume that upon the initial connection of a new player to the gaming infrastructure, the client application is connected to the server with the best tradeoff between network performance and server load, among the available ones. Many policies could be defined to realize a ranking evaluation of available servers. Unfortunately, due to the highly dynamic characteristics of server loads and network performances, any optimal initial allocation would soon become sub-optimal. As an example, network loads, link utilization and router congestion over generalized multi-purpose networks are subject to fast and unpredictable changes, so that any optimal choice at time t may become suboptimal after few seconds. In practice, our proposal is to let the clients, being connected to one server, to *migrate* towards the "most performant" server evaluated at runtime. This migration should be evaluated on the basis of server-measurement metrics and heuristic policies. Also, the density of migrations should be controlled, as an example, by performing migrations only at predefined time-interval boundaries, or after some critical events occurred. In this way, any client should pursue the most performant server, by dynamically measuring the network and server performances. The performance estimates and the migration heuristic should be defined in order to avoid overheads and useless fluctuations of clients.

This paper is structured as follows. In section II we will describe a general architecture for online massive real-time multiplayer games, named *MMORPGs*, or *Massive Multiplayer Online Role-Playing Games* [11, pp. 1–2]. We illustrate the implementation and the behavior of the proposed gaming architecture, together with the fairness problems that may be obtained. In section III we sketch the model of the proposed architecture, and we define the set of parameters that will be adopted in the model to characterize the gaming and the network issues used in the simulations. In section IV we will discuss the implementation of the *client migration*, devoted to the improvement of the game fairness issues. In section V we will present the experimental results of our simulation tests. Different implementations of the migration scheme and heuristics will be discussed and evaluated. Conclusions and future works will conclude the paper.

II. Online Gaming Architectures

Big multiplayer gaming systems are usually based on a pool of server geographically distributed across one or more countries [12]. These servers share the knowledge about the "simulated world" where the characters move and act. When a user wants to play the game in multiplayer mode, he/she is required to connect to one of the available servers, where the game activity is currently on, then he/she starts playing. The choice of the server is usually made by following two possible implementations: *i)* a round-robin DNS entry or *ii)* the "nearest server" selection. The concept of "near" server is usually intended as the "geographically nearest" to the client. Both of these solutions are not guaranteed to provide the best performances. Another solution is to connect to the "topologically nearest" server, i.e. the server reachable by traversing the minimum number of intermediate routers within the network structure. Additional advantage could be given by taking into account the congestion along the path to connect to the server.

Our goal is to analyze how the communication between a client and one of the servers could be improved without touching the network communication protocols, and by creating an additional, external layer with respect to the gaming infrastructure [13]. Specifically, the class of games we are interested in contains the *Massive Multiplayer Online Role-Playing Games* (*MMORPGs*) [11, pp. 1–2]. This class of games is becoming more and more popular on the Internet, and includes multiplayer games that simulate a "virtual world". A great number of characters, either under human or computer control, live and act in the virtual world, performing some cooperative or competitive tasks. MMORPGs are the computer-based counterpart of the well-known RPGs (Role Playing Games) [14]. Each new player creates its own *avatar* with a set of characteristics (e.g. strength, intelligence, initial resources) and starts playing the avatar life in the simulated world, by interacting with other players in order to complete tasks, to compete with enemies, to collect shared and available resources, and to improve the characteristics and skills of his avatar.

MMORPGs may involve a great number of concurrent players. While in other games, like strategic and first-person-shooters [15], [16], the maximum number of interacting players seldom exceeds a dozen [17], MMORPG servers usually have to manage hundreds, or even thousands of avatars co-existing in the same virtual world [18].

A. MMORPGs Gaming Architecture

The typical MMORPGs gaming structure is a simple client-server architecture: each player runs a client, which connects to one out of a pool of many servers. The client and the server start to exchange gaming data as long as the user plays, then the client disconnects. Basically, the gaming protocol governing the system evolution is a simple request/reply scheme: clients send commands to one server, and servers synchronize and send "replies" to the clients. A client/server connection starts when a client sends a login request to one of the servers. In our simulations, we implemented a round-robin DNS entry selection of the server to connect to. When a server receives a login request, it performs a service admission control, to admit a limited maximum number of clients, and then it sends a reply, either an *accept* or a *reject* message to the client. Upon a negative answer, the client will retry with another server in round robin fashion until a server accepts it. After a successful "login" the client starts sending play requests that will be received by the server, enqueued and processed on the next timestep boundary. After having processed a request, the server replies to the client by sending him the updated information about the "world" evolution. This communication and synchronization process goes on until the client stops playing. When the client stops playing, it sends a logout request to the server, it waits for an acknowledgment, and then it quits the game. To maintain full synchronization for inactive players, servers send proactively updates to every client in every timestep. The implemented server synchronization protocol is trivial: each server periodically sends the system status updates to every other server.

Our assumptions include that the servers are connected to each other by high speed links. Every server maintains a replica of the simulated world where the locally managed subset of avatars evolves by executing the player's commands. Every player continuously sends unicast messages to its server, while servers send update information to every other server in order to manage the time synchronization and updates of the virtual environment. This is an inefficient way [19] to handle synchronization, and state sharing, in particular when the number of servers is high. The number of messages among servers grows as $\Theta(n^2)$ where n is the number of servers. These system (and modeling) assumptions for this work were selected in order to consider the most general scenario, and a "worst case" implementation that would be highly affected by the distributed communication bottlenecks. It is worth noting that this choice about the server-to-server communication would not affect positively the results of our migration mechanism. Also, in our model we do not depend on a specific network topology.

B. Time Management and Fairness

We define as Virtual Time (VT) the time in the virtual world, as Simulation Time (ST) the time concept considered for synchronization issues in the distributed simulation, and as Wall Clock Time (WCT) the time we are living [20, pp. 27– 30]. The VT is defined by discrete *steps* of fixed duration (typically in the order of tenths of a WCT second). During each VT step a client can perform at most one *move*, that is, a single command can be sent from the client to the server. With this system (and modeling) assumption, we can control the maximum speed of every player under a common upper bounded speed limit. Every command and update received during step $t-1$ is enqueued on the server managing the avatar, and it will be processed at the beginning of the step t in FIFO order, by causing an update of the virtual environment. The timestepped approach for time management introduces a controlled upper bounded delay for the execution of commands. This can be exploited to introduce a positive fairness principle for clients' commands. The fairness can be improved because fast clients can be limited to perform only one move per timestep, and the timestep size can be tuned such that the same opportunity can be given, on the average, to the majority of slow clients. In this way, all the clients achieve more chances to fairly compete under the remote control by their respective players. Obviously, the timestep duration is a tuning parameter and should not be excessive, in order to maintain a smoothed game evolution. In addition, a MMORPG presents the same issues and the same concept of a real-time simulation. Like in real time simulations, under the interaction viewpoint, the speed gap between the VT of the avatars and the WCT of the players must be negligible.

III. The Gaming Architecture Model

In this section we describe a simple and generic model for the architecture of a massive online multiplayer gaming architecture. The target of our modeling and simulation investigation will be a qualitative comparison between a "classic" gaming and network infrastructure scenario and the same scenario with the effects of the migration mechanism. The definition of an accurate network model or a specific network infrastructure, is out of scope for our goal, because it would be difficult to design and would result in less general results [21]. For the same reasons, we do not aim to define an accurate model of a specific game, and its architecture, but we prefer to characterize a class of online interactive games in general. On the other hand, we would like to capture the network and gaming infrastructure dynamics and characteristics in our model, in order to obtain significant results. To this end, we performed an accurate selection of modeling parameters and their respective values and distributions.

A. The Model Parameters

The communication protocol at the basis of our architecture has been developed after the accurate analysis of network traffic traces obtained by different real games [3], [5], [22]–[24]. Since most of these games are commercial and proprietary we could not examine the protocol details. The only possibility we have is to capture and analyze the real network traffic generated, to infer the parameters' values to be adopted for the communication protocols' modeling[1]. It must

[1]We wish to thank for their help the system administrators of Midian and Nexus gaming networks' infrastructures [25], [26]

be cleared that the traffic model obtained is characterizing the considered gaming application in isolation. Additional network traffic generated by concurrent applications in real systems would increase the network congestion and the communication bottlenecks. This fact is not critical for our analysis and results, since we are interested in qualitative comparison of alternative solutions, under the same assumptions.

B. The Simulated Network Model

The focus of the network modeling is to determine transmission times for the packets, over generalized network infrastructures (inspired by the Internet).

As in real scenarios, we model a packet based network communication based on a standard TCP/IP Internet protocol architecture. On the other hand, we do not include a detailed management of transmission failures and packet retransmissions. The network is assumed to be reliable, since the scope of our analysis does not include fault-tolerance, routing and link failure issues [27]. We assume that communication paths are reliable and we model the effect of network congestion as a variable network latency parameter. This latency effect represents the only overhead effect of network congestion that we include in our model. Packet loss will be modeled as consequence effect of communication delays. By relaxing most of network details, we assumed that transmission times are mainly determined by *link bandwidth* and *link latency* network parameters [21]. We assume the network is organized in a numbered set of A *areas*, roughly corresponding to big Autonomous Systems, Internet Service Provider networks or Internet carriers (see Fig. 1).

A parameterized matrix L is defined, such that each pair of network areas, e.g. $(i, j), 0 \leq i, j \leq A$, is assigned a *base latency* $L_{i,j}$. This base latency value parametrizes a right-shifted lognormal distribution. This distribution is adopted by a pseudo-random generator, to obtain synthetic samples of the latency values for simulated packets traveling from area i to area j and vice versa. The matrix is configured with low base latency values when $i = j$ (i.e. when the two hosts belong to the same area). The right-shifted distribution offset is motivated by the characteristics of the link technology, introducing a "lower bound" for their respective latencies. Dynamic congestion can be modeled at runtime during the simulation by varying the base latencies values in the matrix L. Each client and server host is assigned a *connection class* that identifies the type of connection the host adopts (i.e. analog modem or DSL for clients, various kinds of ATM links for servers). Each connection class corresponds to a nominal *bandwidth* value, affecting the speed of packets sent and received by that host h. This parameter is also used to define a latency multiplier factor K_h that will be used to determine the network latencies. Both servers and clients are distributed in uniform way in the network areas. When one host (h_i) belonging to area i sends a packet to another host (h_j) belonging to area j, we assume that this packet will follow a path whose end-to-end latency can be modeled as follows. The end-to-end latency is calculated as a summation

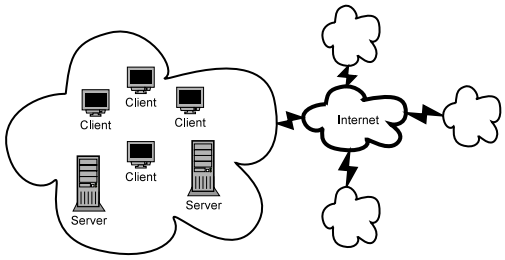

Fig. 1. An example of the network areas

of "intra-area" and "extra-area" latencies and transfer time. Transfer time is defined as the packet size divided by the link bandwidth of the slowest link in the path, with no variability introduced by Medium Access Control and Data Link policies. As an example, a packet sent with an analog modem comes out from the modem at 33 Kbps, no matter how much fast it will travel in the rest of the network.

If the hosts h_{i1} and h_{i2} belong to the same area i, we call the end-to-end latency as "intra-area latency". It results that "intra-area base latencies" can be found on the diagonal of the matrix. In this scenario, the end-to-end latency is determined as a composition of two expressions:

$$\begin{aligned} Latency(h_{i1}, K_{h_{i1}}, h_{i2}, K_{h_{i2}}) = \\ K_{h_{i1}} * L_{i,i} + K_{h_{i2}} * L_{i,i} \end{aligned} \tag{1}$$

where $K_{h_{i1}} * L_{i,i}$ is the initial "intra-area latency" of host h_{i1}, and $K_{h_{i2}} * L_{i,i}$ is the final "intra-area latency" of the destination host h_{i2}, $\forall h_{i1}, h_{i2} \in i$.

If h_i and h_j belong to different areas, the overall latency is determined as a composition of three expressions:

$$\begin{aligned} Latency(h_i, K_{h_i}, h_j, K_{h_j}) = \\ K_{h_i} * L_{i,i} + L_{i,j} + K_{h_j} * L_{j,j}, \qquad (i \neq j) \end{aligned} \tag{2}$$

where $K_{h_i} * L_{i,i}$ is the initial "intra-area latency" of area i, L_{ij} is the intermediate "extra-area latency" which is the latency of the network connecting the originating and destination areas, and $K_{h_j} * L_{j,j}$ is the final "intra-area latency" of the destination area j.

The choice of the model parameter values (distribution parameters [28], [29], connection classes characteristics [30], percentage of connection classes) has been made *i)* with direct tests and traffic analysis of real network connections [22], [23], *ii)* by gathering data from some Italian ISPs, carriers and important web sites, and *iii)* by collecting information from other papers and articles.

In Figs. 2 and 3 we show the experimental ping-time (latency) results obtained when investigating the latency distribution inside the GARR network, and between the GARR network and the Teleglobe network (belonging to two different autonomous systems). This shows that the typical latency distributions for ping times, both inside the same area (Fig. 2) and across two or more areas (Fig. 3), could be approximated by a right-shifted lognormal distribution [32], [33].

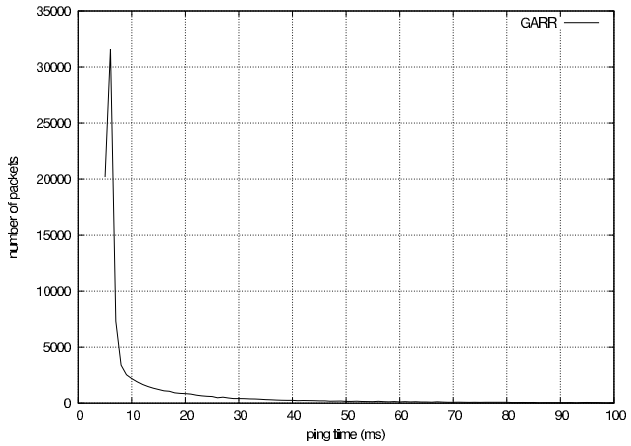

Fig. 2. Latency inside the GARR Network [31]

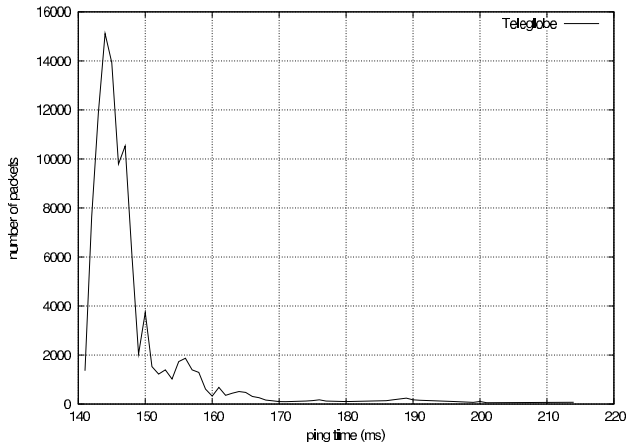

Fig. 3. Latency between the GARR Network and the Teleglobe Network

The distribution of the various connection classes for carriers and big ISPs has been determined for our model, thanks to information from the Italian Internet Exchange [34]. The resulting information about link-types, link-bandwidths and percentage of "presence" values are reported in table I.

The distribution of the various connection classes for gaming clients has been determined for our model, thanks to information from the Midian gaming network [25]. The resulting information about line-types, upstream and downstream line-bandwidths, connection class latency multipliers (K_h) and the percentage of "presence" values are reported in table II.

To complete the network model parameterization we analyzed the traffic generated by real games with the aim of discovering information about the packets size distribution and packets' correlation. The proprietary protocols and packet formats used by gaming platforms hidden all details, hence our observation was based on network monitoring. We discovered that while clients send out few different types of packets, a server generates variable sized packets. Anyway, the typical

Name	Bandwidth (KB/sec)	Presence
DS3	44,736	30%
STS-1	51,840	10%
STS-2	103,680	20%
STS-3	155,520	10%
STS-6	311,040	10%
STS-9	466,560	10%
STS-24	1,244,160	10%

TABLE I
MODEL PARAMETER DISTRIBUTION OF CARRIERS AND BIG ISPS

	Up (KB/sec)	Down (KB/sec)	Latency multip.	Presence
POTS 36.6K	27	27	5.0	4%
POTS 56K	40	25	4.0	24%
ISDN 64K	64	64	3.0	20%
ISDN 128K	128	128	3.0	6%
ADSL 256K	128	256	2.0	20%
ADSL 640K	128	640	2.0	14%
Optical fiber	10000	10000	2.0	12%

TABLE II
MODEL PARAMETERS DISTRIBUTION OF GAMING CLIENTS

packet size is 42 bytes. Hence, we considered this value as the standard play request/reply packet size, and we used different sizes for other types of event messages.

C. The Server Model

While we found good data and references about traffic and network characterization of modeling parameters [17], [24], [35], we had some difficulties in finding concrete values for the server computation model parameters (e.g. CPU times). Hence, to characterize the CPU computation time distributions of the servers we executed external observations, and we measured the time passed from the arrival of a request to the server and the departure of the reply.

IV. THE MIGRATION MECHANISM

In classical gaming implementations, each client chooses its server at the beginning of its playing, and keeps using it until the end of the playing session. Currently, there is no information to assist the client in making a "good choice" at the beginning of the game session. For this reason we suggest the introduction of a *client migration mechanism* in our architecture. The migration mechanism would allow a client to disconnect from a server, and to connect to another server, while continuing to play in user's transparent way. The migration mechanism should be activated when a client "realizes" that the server connection is slow, or another server is reachable by means of a more fast connection. In this way a better exploitation of the overlay network of server replicas could be performed. One critical point about the migration mechanism is the design and implementation of the migration heuristics to assist clients in the decision to migrate, and also

in the selection of a new destination server. In the following we describe two approaches that we considered in our analysis.

As we said before, the communication between a client and a server is affected by two important factors: bandwidth and latency. From the client viewpoint, the bandwidth factor has less influence over the game performances: a high percentage of the packets are quite small [36] so that they can be sent on a low bandwidth link without saturating it. Anyway, the effect of a low bandwidth link can be mitigated by sending less detailed information to the players on that link. The most critical parameter we are dealing with is the end-to-end (E2E) network latency. The E2E latency is a composition of delays introduced by single network links realizing a path from the client to the server and vice versa. Most of the times, the latency of the first hop (usually from the client to the router of its area boundary) cannot be negotiated or controlled by the client. On the other hand, the client can control the game server to connect to, and this allows the client to have many possible alternative paths with different latency performances to equivalent servers. While the first communication step usually has quite low latency variance, the path between the originating ISP and the remote server can have much less predictable latencies due to external Internet traffic aggregation. Due to the network latency variations, the initial choice of the optimal server could not be valid indefinitely. To maintain the optimality, a runtime adaptive selection and migration to adapt to the network conditions is preferable. The client migration can improve the fairness, and can be applied with a limited amount of work on the existing game code, while other methods would require relevant changes to the protocols [9], [37].

Big software houses that create and distribute online multiplayer games consider the network latency a serious problem, such that they manage and invest in servers and networks to offer their customers good network performances and to be competitive with other game producers. Other gaming networks [38]–[40] are independent to software houses that produce games, and appear to be quite open to publish technical details about their network structures and protocols.

To maintain the E2E latency upper bounded, many gaming infrastructures, and specifically MMORPGs infrastructures, consider the network partitioned in different wide *areas*, covering whole countries or group of countries [5]. Unfortunately, to maintain massive and global gaming infrastructures, areas can be wide, including multiple servers that appear with different performances to clients. This work's in favour of adopting a migration algorithm, for the motivation above.

Intuitively, the aim of the migration mechanism is to make the latency curve distribution of the clients as much homogeneous as possible. This means that the game service is fairly available, because the latency of a client is similar to the latency of other clients. A timestepped time management like the one defined in previous section could additionally play in favour of the fairness and latency clustering-effect of clients, by letting "fast" clients to be synchronized to the rhythm sustainable by "slow" clients.

A. The Migration Heuristics

When a client determines it should try to move to another server, it has to decide which one is the "best choice" for its new connection. As said before, the link bandwidth is not a critical factor, and cannot be modified, because the slowest network segment usually is the one which connects a user to his ISP.[2]

A good migration heuristic should consider at least two other factors: the *network latency* between the client and the server, and the *server load*. We assume that each client performs periodic pings towards every possible candidate server, to obtain dynamic estimates of the network latencies. This method is simple, it gives quite a good approximation of the "real" latency and allows a experimental validation of assumptions. Moreover, the ping rate allows to control the overheads introduced. The pitfall of the ping method is that a well connected server could result the target for many clients connections, and this may saturate the server capacity. In the proposed architecture, we propose a simple load balancing policy based on service admission principles: a server keeps track of the number of currently connected clients and refuses to accept a new connection if its load becomes too high (causing a global performance loss). In future improvements of this architecture, servers could exchange load indicators, by defining a new dynamic distributed load-balancing technique.

1) The migration trigger: A very important point is the algorithm that activates a client migration. We mainly focused on the simplest of all, named *simple migration*, and in addition, we made some tests on a more complex algorithm that we called *early migration*.

2) Simple migration: The *simple migration* algorithm is the following: once every a fixed number of seconds the client sends a ping probe (ICMP ECHO REQUESTs) to the servers in the list. If a different server appears to be faster than the current one, then a migration process will be activated. Otherwise, no client migration is performed. This policy is simple and stable, because the frequency of migrations can be tuned by paying some reactivity to network dynamics.

3) Early migration: The *early migration* evaluates migration opportunities at fixed time intervals, like previous policy. In addition, clients collects runtime statistics on the network round trip time by exploiting commands/replies from/to the server. If the runtime statistics indicate that the current connection is becoming too slow, then the client should try to *anticipate* the migration attempt. Many possible implementations can be obtained by considering different statistics and thresholds.

B. The Migration Protocol

The client starts from the first server of the list, and if the migration is considered useful, it sends a packet to its current server by asking him to migrate to a new one. Upon migration request, the current server takes care of contacting the new

[2]For instance, a user connecting with a 56Kbps modem is mainly limited by this bandwidth bottleneck, not by his server's one.

server. If the new server has sufficient available resources for a new client, the current server sends the client data to the new server. When the new client instance is active on the new server the client is notified and starts sending packets to the new server. Meanwhile, the old server forwards the received packets to the new server and routes back the replies. In this way the migration process is transparent to the client that keeps playing during the migration with no additional delays. If a server sends a negative reply to a migration request, the client asks for the following server in its list, until the end of the list is reached, or the migration is completed. The migration protocol is simple, and can be implemented as a logout/login sequence with some intermediate steps to move the client status data and the packet forwarding during the migration phase. The key assumption is that every server maintains the information of managed clients only, together with the information of the whole virtual environment, so that no environment information needs to be migrated. The migration implementation may be dependent to the game platform. Anyway, our preliminary considerations over generalized gaming architectures shown that dependencies from the core of a gaming system should not require complex adaptation efforts.

V. Experimental results

A. Simulation Tool

In order to simulate the proposed migration-based architecture, we do not use any existing simulation tool or runtime, but we created a dedicated simulator, opportunely optimized for the RAM management, in order to overcome the memory bottleneck in a massive simulation [18]. Another reason for this simulator choice was the model simplicity and simulator speed. Existing powerful runtime environments (e.g. [41]), would have required a complex management and modeling efforts not motivated by the need for high simulator potential. Given our assumptions, the simulation model is quite simple, it is highly modular, and does not require complex simulation services. We want to simulate MMORPG networks with thousands clients connected at the same time, where each client communicates with at least one server and each server communicates with every other server. We implemented, tested and validated a dedicated event-driven simulator written in C language, with good performances and low memory requirements. The discrete event-based simulation paradigm was preferred to a time stepped simulation [42, pp. 7–9, 93, 94] mainly because the behavior of thousands simulated objects generating few events is more efficient to be simulated via discrete event-based simulation.

B. Simulation setup

The simulation parameters have been discussed in previous sections. Specifically, we defined the values of the random distributions parameters and their types, the percentage of clients and servers of each connection class, the size distribution of variable kinds of packets [36], the bandwidth and latency characteristics of common network connection classes, and the CPU time required to manage play requests.

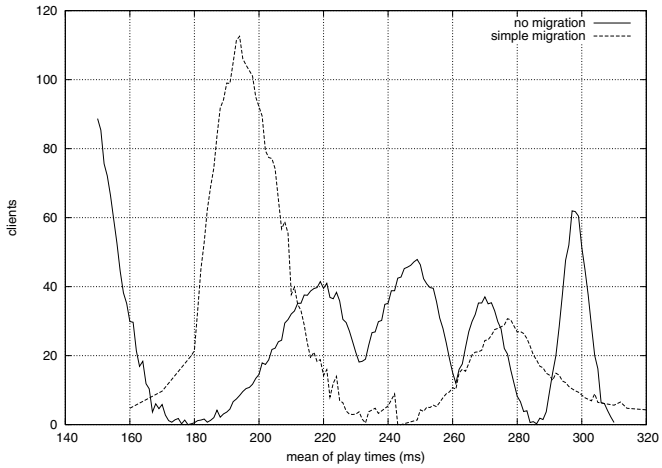

Fig. 4. Distribution of clients over average play time values: simple migration vs. no migration

Our typical simulated scenario is composed by a network with 20 gaming servers and 2000 concurrent clients. All the simulated scenarios' parameters, like the number of clients and servers, the time each client spends in a connection and the frequency it sends play requests have been chosen in order to reflect a real scenario. A number ranging from 10 to 20 independent runs has been executed for each test in order to achieve a good confidence level [42, pp. 253–259].

C. Performance Results

The main performance index will be referred to as *play time*, which is defined as the time a client has to wait for a reply after it has sent a request. The play time is expressed in milliseconds (ms) and it is used to compare the *fairness* of a gaming infrastructure under many different scenarios.

The first graph (Fig. 4) is a comparison between the scenario characterized by the original MMORPGs architecture and the same architecture with the simple migration algorithm (see section IV). The figure shows the distribution of clients (Y axis) with respect to their respective average play time values (on the X axis). The solid line represents the original architecture, while the dashed line illustrates the effect of the "simple migration" mechanism over the same scenario. With the original architecture, the play times appear to be clustered around at least 5 sets of distant local aggregation values. This demonstrates that clients would fall with a certain distribution in groups characterized by different play times performances, resulting in unfair gaming service. The play times appear to be much more concentrated with the simple migration mechanism. There is a small group in the right part of the graph that depends on the clients with very slow connections, and another big group on the left with really good average performances. It is worth noting that, in the migration-enabled system, a timestep around 220 ms would satisfy in fair way a majority of clients, with respect to the classical implementation.

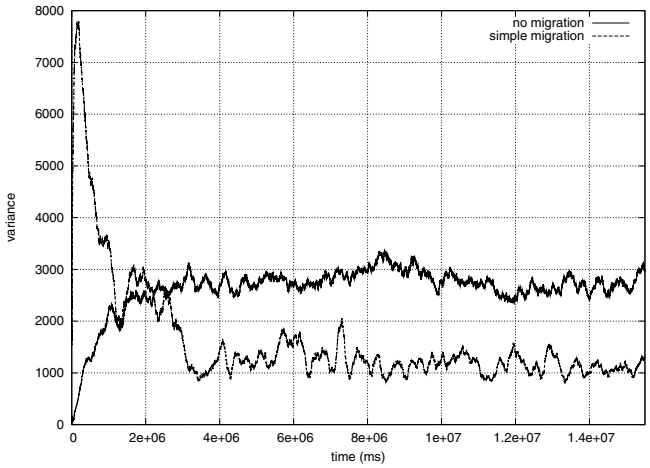

Fig. 5. Dynamic estimated variance of the play time values (ms)

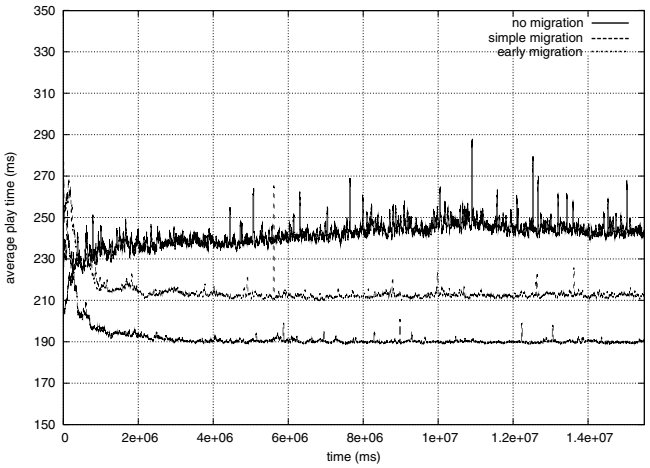

Fig. 6. Runtime average play time values (ms)

In Fig. 5 we show the runtime evolution of a fairness index. We think that our concept of "fairness" may be represented as the *variance* of the average play times of the clients in a given time interval (1 s). If the variance value is low, each player experiences a latency which is similar, on the average, to the others players'. On the other hand, a big variance value means that the play times are much more heterogeneous, hence some players suffer the unfairness of the gaming infrastructure. The original architecture (solid line) leads to higher variance value than the same architecture with the simple migration. After an initial transient phase, the migration-enabled architecture outperforms the classic architecture in terms of fairness. In table III, we calculated the confidence intervals for the estimated play time variance. The narrow interval obtained by the simple migration mechanism confirms the expectations about the generalized and uniform level of fairness obtained by the clients of the gaming architecture. Quite surprisingly, the confidence interval for the fairness index obtained with the "early migration" heuristic mechanism is larger than the same index in the original architecture with no migration. This was due to the fluctuating behavior of subsets of clients swinging between two servers, by introducing migration overheads and unbalanced loads on the servers.

The analysis on the *average* value of the play times shows that the average time for a reply, i.e. the system interactivity, is favoured by the introduction of the simple migration. This can be explained, as we can see from the Fig. 4, even if some clients are subject to penalty (i.e. the first high spike in the distribution on the left of the solid line is shifted to the right in the dashed one). The penalty for fast clients is quite limited and was considered as a consequence of our design, given the initial assumptions. This fact is not critical, because it does not compromise the interactive potential of fast clients. As a good news, this flattens the differences of play times between fast and slow clients.

In Fig. 6 we show the average play time of messages sent by all the clients in the simulated system during a given time interval. It is clear that at runtime the "simple migration" mechanism gives the better results, with respect to the "early migration" scheme, which in turn outperforms the classical static (no migration) implementation.

Fig. 7 shows the same index of Fig. 6, but this time the X axis is given by the number of moves performed, which is related to the player life duration in the game. The X axis shows the numbered sequence of requests of each client, and the Y axis reports the average play time of the i^{th} request by the set of clients that were involved in the simulation. This performance index needs to introduce some preliminary considerations. First, this index is correct, since the rate of commands sent by slow and fast clients is homogeneous and comparable. Second, each client remains connected for a certain amount of time that has a pre-defined average value [43]. There are many motivations that justify this choice in the model parameterization: mainly, slow clients are usually modem connected, hence they pay for time of connection, and, also, they may experience sudden death of their avatars due to connection delays. The average number of messages sent by slow clients falls around the middle of the X range in the figure, while fast clients perform longer playing activities. The solid line refers to average play times in the architecture with no migration. With this scenario, the average client has a play time of about 250 ms at the beginning of the playing interval, and this index appears to decrease when the slow clients start to abandon the game. After $X = 8000$ moves in the figure, the survived players in the game are mostly fast clients. This is shown in the figure because average play time starts to decrease.

The dashed line shows the average client play times in the "simple migration" scenario. The average play time drops to a low play time after the first migration assessment occurs. The fact that the dashed line converges to a constant value (instead of decreasing like with the solid line) means that every client experiences a comparable play time, which is maintained also when slow clients (e.g. modem enabled) start to leave the

	No migration	Simple migration	Early migration
Run replicas	10	10	10
Run samples	80,221	63,013	62,610
Transient samples	12,000	12,000	12,000
Average Play Time (ms)	2,816.69	1,203.03	1,664.48
Estimation of σ^2	4,766,163.53	965,389.76	4,809,603.78
Play Time Confidence int.	**1,551.23 — 4,082.14**	**633.51 — 1,772.56**	**393.27 — 2,935.69**
Play Time Conf. Interval width	2530.91	1139.05	2542.42

TABLE III

CONFIDENCE INTERVALS OF PLAY TIME VARIANCE (90-TH PERCENT CONFIDENCE LEVEL)

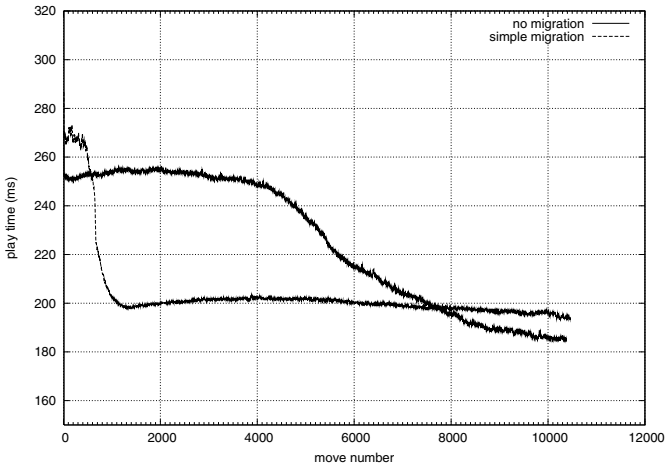

Fig. 7. Average of play time values move by move

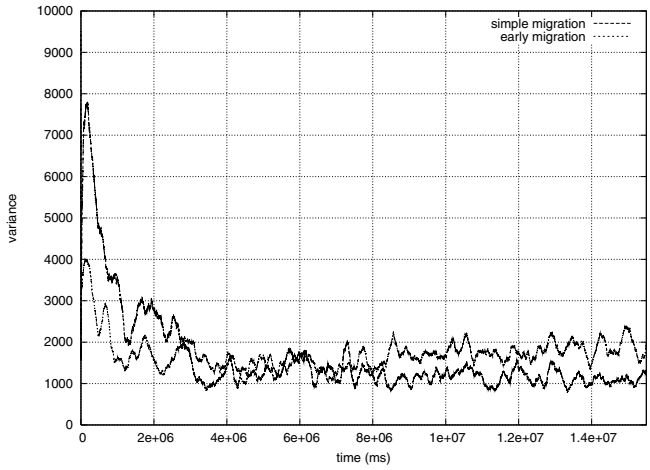

Fig. 8. Dynamic estimated variance of the play time values (ms), simple vs early migration

game, and this confirms the improvement in fairness.

The fact that the average play time with "no migration" outperforms the play time with "simple migration" after $X = 8000$ moves can be explained because in the "no migration" scenario most of the slow clients already left the system and the reduced number of clients increases the servers and network performances.

The "early migration" heuristic has been evaluated with the same tests performed for the "simple migration" and no migration mechanisms. The results shown that the "early migration" scheme have a worst behavior than the simple migration. A comparison between the two types of migration algorithms is shown in Figs. 8 and 9. The dashed line is for the simple migration, the dotted one is for the early migration. The analysis of simulation traces shown that the early migration scheme has some difficulties when there are two or more servers with a very similar latency. This makes a subset of clients to swing between two servers, by introducing migration overhead and unbalanced loads on the servers.

VI. CONCLUSION AND FUTURE WORK

In this work, we presented an improved architecture model for online multiplayer games, focused on MMORPGs with multiple replicated servers. We designed and implemented a simulator and a model for the gaming infrastructure and for a general network used by clients and servers for their communications. Over the proposed gaming architecture we realized the preliminary design of a client migration protocol and a couple of client migration heuristics. The migration mechanism realizes a low cost dynamic adaptation of the gaming architecture to the limitations of the communication infrastructure. The proposed mechanism has been simulated over the class of distributed multiplayer and multi-server interactive games, implemented over a general communication network infrastructure. Results show a significant fairness improvement, more homogeneous performances, and the absence of significant overheads, with less or no modifications required to the gaming architecture.

Additional studies will be performed to extend the client migration mechanism: more efficient migration algorithms, new heuristics tuned on new runtime statistics, and the refinement of communication protocols between servers. The creation and design of gaming architectures with "database server" for the storage of player information, and "area servers" for the elaboration of events in different areas of the simulated world [44] will be evaluated. The gaming architecture will be integrated by a pool of "proxies" accepting connections from the clients and forwarding their requests to the right area server [13], [45]. We will also investigate the possible decoupling

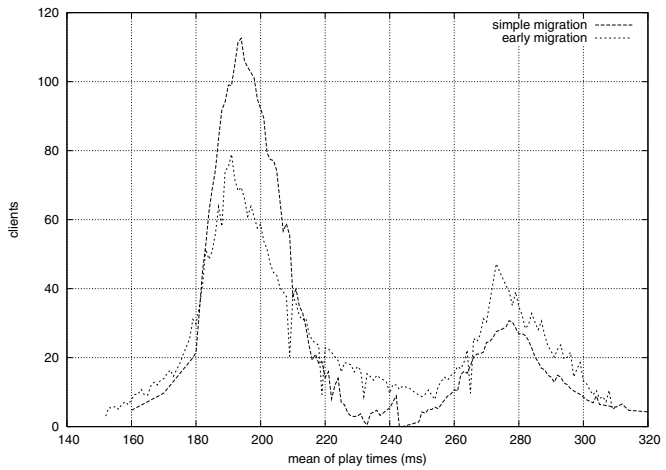

Fig. 9. Distribution of clients over average play time values, simple vs early migration

and integration of the client migration and the remote and replicated server execution.

More detailed and complete network and gaming architecture models will be completed and refined for simulation purposes.

ACKNOWLEDGMENT

This work is supported by MIUR FIRB funds, under the project: *Performance Evaluation of Complex Systems: Techniques, Methodologies and Tools*.

REFERENCES

[1] Blizzard Entertainment, "Warcraft 3," http://www.blizzard.com/war3.
[2] P. J. Unold, C. L. Gregersen, A. O. Kjeldbjerg, *et al.*, "Freeciv," http://www.freeciv.org.
[3] Bioware Corp., "Neverwinter night," http://nwn.bioware.com.
[4] Gravity Corporation, "Ragnarok online," http://iro.ragnarokonline.com.
[5] Origin, "Ultima online," http://www.ultimaonline.com.
[6] M. Oliveira and T. Henderson, "What online gamers really think of the internet?" in *Proc. of the 2nd workshop on Network and system support for games*. New York, NY, USA: ACM Press, 2003, pp. 185–193.
[7] T. Henderson, "Latency and user behaviour on a multiplayer games server," in *Proc. of the 3rd Intl. Workshop on Networked Group Communication (NGC)*, November 2001, pp. 1–13.
[8] J. Smed, T. Kaukoranta, and H. Hakonen, "A review on networking and multiplayer computer games," http://citeseer.nj.nec.com/article/smed02review.html, Turku Centre for Computer Science, Tech. Rep. 454, Apr. 2002.
[9] S. K. Singhal and D. R. Cheriton, "Using a position history-based protocol for distributed object visualization," http://citeseer.nj.nec.com/119175.html, Tech. Rep. CS-TR-94-1505, 1994.
[10] L. Gautier and C. Diot, "Design and evaluation of mimaze, a multiplayer game on the internet," in *Intl. Conf. on Multimedia Computing and Systems*, 1998, pp. 233–236.
[11] J. Smed, T. Kaukoranta, and H. Hakonen, "Aspects of networking in multiplayer computer games," in *Proc. of Intl. Conf. on Application and Development of Computer Games in the 21st Century*, L. W. Sing, W. H. Man, and W. Wai, Eds., Hong Kong SAR, China, Nov. 2001, pp. 74–81.
[12] W. chang Feng and W. chi Feng, "On the geographic distribution of online game servers and players," in *Proc. of the 2nd workshop on Network and system support for games*. ACM Press, 2003, pp. 173–179.
[13] M. Mauve, S. Fischer, and J. Widmer, "A generic proxy system for networked computer games," in *Proc. of the 1st workshop on Network and system support for games*. ACM Press, 2002, pp. 25–28.
[14] Wizards of the Coast, Inc., "Dungeons & Dragons," http://www.wizards.com/dnd.
[15] Epic Games, Inc., "Unreal Tournament," http://www.unrealtournament.com, 2004.
[16] Valve Corporation, "Counter–Strike," http://www.counter-strike.net.
[17] W. chang Feng, F. Chang, W. chi Feng, and J. Walpole, "Provisioning online games: a traffic analysis of a busy counter-strike server," in *Proc. of the second ACM SIGCOMM Workshop on Internet measurment*. ACM Press, 2002, pp. 151–156.
[18] H. Engum, J. V. Iversen, and yvind Rein, "Zereal: A semi-realistic simulator of massively multiplayer online games," http://citeseer.nj.nec.com/555870.html.
[19] E. Cronin, B. Filstrup, and A. Kurc, "A distributed multiplayer game server system," http://citeseer.nj.nec.com/cronin01distributed.html.
[20] R. M. Fujimoto, *Parallel and Distributed Simulation Systems*, 1st ed. John Wiley & Sons, Inc., 2000.
[21] S. Ford and V. Paxson, "Difficulties in simulating the internet," *IEEE/ACM Transactions on Networking (TON) archive*, vol. 9, no. 4, pp. 392–403, August 2001 2001.
[22] V. Jacobson, C. Leres, S. McCanne, *et al.*, "Tcpdump," http://www.tcpdump.org.
[23] G. Combs *et al.*, "Ethereal: A network protocol analyzer," http://www.ethereal.com.
[24] R. Bangun and E. Dutkiewicz, "An analysis of multi-player network games traffic," in *Multimedia Signal Processing, 1999 IEEE 3rd Workshop on*, 1999, pp. 3–8.
[25] "Midian server for ultima online," http://midian.gamesnet.it.
[26] "Nexus ultima online server," http://www.nexus.sm.
[27] M. Borella, D. Swider, S. Uludag, and G. Brewster, "Internet Packet Loss: Measurement and Implications for End-to-End QoS," in *Proc. of the Intl. Conf. on Parallel Processing Workshops*, 1998.
[28] J. Bentini, "Architettura distribuita scalabile per applicazioni interattive e multiutente su internet," Master's thesis, Università di Bologna, 06 2003.
[29] J. C. McEachen, "Traffic characteristics of an internet–based multiplayer online game," in *4th Intl. Conf. on Information, Communications & Signal Processing*, 2003.
[30] L. L. Peterson and B. S. Davie, *Computer Network — A Systems Approach*, 2nd ed. Morgan Kaufmann, 2000.
[31] "GARR - The Italian Academic and Research Network," http://www.garr.it/garr-b-home-engl.shtml.
[32] MathWorld, "Continuous Statistical Distributions," http://mathworld.wolfram.com/topics/ContinuousDistributions.html.
[33] T. Lang, G. Armitage, P. Branch, and H.-Y. Choo, "A synthetic traffic model for Half-Life," in *Australian Telecommunications Networks & Applications Conference 2003*, Melbourne, Australia, December 2003.
[34] Milan Internet eXchange, "Charter members," http://www.mix-it.net/elenco_afferenti.html?lang=it&order=banda.
[35] J. Farber, "Network game traffic modelling," in *Proc. of the 1st workshop on Network and system support for games*. New York, NY, USA: ACM Press, 2002, pp. 53–57.
[36] M. Borella, "Source models of network game traffic," *Computer Communications*, vol. 23, no. 4, pp. 403–410, 2000.
[37] Y. Lin, K. Guo, and S. Paul, "Sync-ms: Synchronized messaging service for real-time multi-player distributed games," in *Proeedings of 10th IEEE Intl. Conf. on Network Protocols (ICNP)*, 2002.
[38] "Ngi," http://www.ngi.it.
[39] "Tgmonline," http://www.tgmonline.it/fragzone/server.
[40] "Gamesnet," http://www.gamesnet.it.
[41] J. S. Dahmann, R. M. Fujimoto, and R. M. Weatherly, "The Department of Defense High Level Architecture," in *Winter Simulation Conf.*, 1997, pp. 142–149. [Online]. Available: citeseer.nj.nec.com/dahmann97department.html
[42] A. M. Law and W. D. Kelton, *Simulation Modeling and Analysis*, 3rd ed. McGraw Hill International Series, 2000.
[43] F. Chang and W. chang Feng, "Modeling player session times of on-line games," in *Proc. of the 2nd workshop on Network and system support for games*. ACM Press, 2003, pp. 23–26.
[44] J. Aronson, "Using groupings for networked gaming," http://www.gamasutra.com/features/20000621/aronson_pfv.htm.
[45] C. Griwodz, "State replication for multiplayer games," in *Proc. of the 1st workshop on Network and system support for games*. ACM Press, 2002, pp. 29–35.