AN OPTIMISTIC OBSOLESCENCE-BASED APPROACH TO EVENT SYNCHRONIZATION FOR MASSIVELY MULTIPLAYER ONLINE GAMES

Stefano Ferretti, Marco Roccetti, Claudio E. Palazzi
Department of Computer Science
University of Bologna
Mura Anteo Zamboni 7, 40127 Bologna, Italy
Email: {sferrett, roccetti, cpalazzi}@cs.unibo.it

Abstract

Massively Multiplayer Online Games (MMOGs) are no longer a chimera. Day after day, new exemplars of this application are released, available for free on the Web or on sale. Yet, high lags frequently affect MMOGs annoying customers. This frustrating phenomenon is mainly due to the best-effort service provided by the Internet that is often responsible for congestion and loss of responsiveness. To address this challenging issue, we consider a *mirrored game server architecture* and present a mechanism aimed at accelerating the delivery of game events generated during game sessions. Our approach rests upon the use of an optimistic synchronization mechanism that exploits the semantics of the game to diminish the delivery time of game events, whilst maintaining the full consistency of the game state. Experimental results are reported confirming the adequacy of our approach.

Keywords: Massively Multiplayer Online Games, Mirrored Game Server Architecture, Interactivity, Online Entertainment.

1 Introduction

Since the birth of the game industry, more than thirty years ago, video games have evolved from being a simple pastime for children to a serious business. Nowadays, this form of entertainment represents an important case study for researchers, which have been massively attracted by its technical challenges and sociological implications. Furthermore, since games have become similar to real simulators (e.g., flight-, combat-, vehicle-, human-simulators), other typologies of applications have also converged into game technologies. Indeed, game-based tools can simulate, in a relatively cheap and safe way, situations which could be much more expensive or dangerous in the real world. Examples of applications that may gain benefits from technical solutions introduced by game technology are related to medical surgery, military simulations, interactive collaboration, interactive storytelling, distance learning, e-commerce and manufacturing systems [4, 15, 21, 34, 35, 37]. Summing up, more and more

"non-game applications" adopt elements of computer gaming technology to ameliorate user experiences in several domains [36].

Nowadays, great advances in networking technologies and the wide diffusion of new network-enabled mobile devices are ruling the future of the game industry. In this context, the phenomenon of Massively Multiplayer Online Games (MMOGs) is emerging as a new trend able to involve players connected through the Internet making them interact in a distributed virtual game world. All these factors are leading a steady growth of new game technologies. In essence, the game market is characterized by a rising demand for distributed, responsive, and scalable strategies able to provide players on the net with a feeling of full immersion into the virtual game world during the game escalation.

Several alternative architectures have been proposed to support MMOGs, ranging from client-server [1,14] to peer-to-peer [10]. Embodying an efficient tradeoff among these architectures, *mirrored game server* has recently emerged as a solution able to provide scalability while minimizing the overhead of the game event exchange protocol. In essence, this solution deploys over the network a constellation of replicated *Game State Servers* (*GSSs*), each of which locally maintains a redundant version of the game state. Each *GSS* manages and updates its copy of the game state as follows: i) it collects game events coming both from its engaged players and from other *GSSs*; ii) it forwards to all other *GSSs* the events generated by its connected players; iii) it updates the game state considering the set of received game events; iv) it finally delivers the newly updated game state to its connected players.

It goes without saying that, within this scenario, an efficient event synchronization scheme among *GSSs* needs to be employed to guarantee a consistent and responsive evolution of the game state. Hence, one of the key factors in determining the success of an online game is represented by the ability to rapidly deliver events among the various *GSSs*. Indeed, while simpler turn-based games do not have to face this problem, as only one player is allowed to perform an action at any given time, the task of providing players with responsiveness and real-time interactions is probably the most stringent requirement for MMOGs. In fact, in case of intense traffic in the network or when excessive computational loads are slowing down some *GSSs*, the game delivery activity turns out to be delayed. As a consequence, the responsiveness of the distributed game system may be jeopardized.

With this in mind, a recent study has been proposed where the goal of uplifting the playability degree of MMOGs is achieved by maintaining the game event delivery delays under a

human perceptivity threshold, whilst preserving the game state consistency [14]. At the basis of this approach lies the idea of exploiting the notion of *obsolescence*. Simply put, obsolescence entails that, depending on the semantics of the game, some events can lose their significance as time passes, i.e., new actions may make the previous ones irrelevant. For example, where there is a rapid succession of movements performed by a single agent in a virtual world, the event representing the last destination supersedes the previous ones (obsolete events).

In essence, obsolescence allows the system to drop those game events that lose their importance during the game evolution. Discarding superseded events for processing fresher ones may be of great help for delay-affected *GSSs*. This means that, during the game event exchange activity, while responsive *GSSs* may deliver all game events to provide their connected clients with a fluent game state evolution, those *GSSs* that experience a loss of interactivity may skip the execution of obsolete events in order to speed up the event processing activity, thus gaining interactivity. For example, it might be the case when a delay affected *GSS* p is waiting for the arrival of a game event e_i that is supposed to be processed next. However, if another event e_j, that makes obsolete e_i, arrives at p, then p can quit waiting for e_i and process e_j directly.

Obviously, not every event in a game may be considered as obsolete. Alongside the notion of obsolescence, in fact, is also the notion of *correlation* among events which has to be taken into account. Two events are said to be correlated if the final game state depends on their execution order [14]. In essence, correlated game events often represent significant interactions among game events which require a careful processing in order to obtain a correct evolution of the game state. Thus, obsolescence cannot be applied to correlated game events.

Another interesting effect of correlation amounts to the fact that only correlated events are required to be processed in the same order at all the *GSSs* to maintain the consistency of the game state (*correlation-based order*). Instead, different delivery orders are possible when a given sequence of non-correlated events has to be delivered to *GSSs*.

Based on the notions of obsolescence and correlation, several schemes have been recently proposed that exploit these two properties to improve interactivity [13,14,15,30,31,32]. Among these, some exploit a reactive strategy that monitors the interactivity degree provided by the system. As soon as the measured interactivity degree decreases below a given game perceptivity threshold, an interactivity restoring procedure is activated. This procedure skips processing obsolete game events with the aim of bringing back the measured interactivity level below the

game perceptivity threshold (*Interactivity Restoring*) [14]. Since only obsolete events are discarded at some given *GSS*, this stabilization mechanism reduces the delivery latencies of game events, thus gaining interactivity, without causing inconsistencies in the game evolution.

Alternatively, taking inspiration from the RED (Random Early Detection) approach exploited in the field of congestion control for computer networks [16], another strategy exists that proactively avoids any loss of interactivity before it might happen. In essence, obsolete events are discarded in advance based on a given dropping probability which depends on the measured values of the interactivity degree (*Interactivity-Loss Avoidance*) [30, 32].

However, the problem with all these strategies is that they have been developed based on a conservative approach [17]. Simply stated, the decision at a given *GSS* to process an event e_i is delayed until all non-obsolete events preceding e_i are received by that *GSS*. Hereinafter, we refer to these synchronization mechanisms as *Conservative Obsolescence-based Synchronization* (COS) schemes.

In this paper, we present a novel game event synchronization mechanism for mirrored game servers that exploits the notions of obsolescence and correlation following an optimistic synchronization approach. We refer to our mechanism as an *Optimistic Obsolescence-based Synchronization* (OOS) mechanism. Following our OOS mechanism, every time a new game event e_i is received at a *GSS*, an immediate check for detecting the obsolescence of e_i is carried out. If this check succeeds, the event e_i is immediately discarded. Otherwise, another check is performed to verify whether later events e_j exist that are correlated to e_i and have already been processed. In this case, a rollback procedure is performed as follows: all the aforementioned events e_j are rolled back and a new game state is computed to be now safely delivered to all connected players. Obviously, during the rollback procedure some of these events e_j may become obsolete since new values of the game state variables are computed. In such a case, these obsolete events are simply dropped.

It is widely accepted that the main advantage of optimistic strategies, w.r.t. conservative ones, amounts to the fact that they perform better in fast paced games where a continuous rate of the game advancement is important [8]. The experimental study we have conducted confirms this consideration, especially when the attention is focused on interactivity. In particular, we report results showing that our OOS strategy outperforms alternative COS mechanisms in terms of interactivity as it speeds up the event processing activity at each *GSS*.

The remainder of this paper is organized as follows. An analysis of related works proposed

in literature is presented in Section 2. Section 3 introduces the adopted system model. Section 4 presents our OOS strategy for MMOGs. In Section 5 we report experimental results obtained from a simulative assessment that confirm the efficacy of our approach. We conclude the paper with some final comments in Section 6.

2 Related Work

Aimed at guaranteeing a uniform view of the game state among all *GSSs*, several synchronization schemes have been presented in literature for the support of networked multiplayer games [8,9,10,22,26,27]. Among these, similar to our OOS scheme, some of them take inspiration from the optimistic Time Warp algorithm to adopt a detect-and-correct strategy [8,9,26,27].

A new synchronization mechanism for online games called *trailing state synchronization* was presented by *Cronin et al.* [8, 9]. According to this approach, every *GSS* locally maintains a fixed number of copies of the game state, each of which is kept at a different simulation time. In essence, each copy of the game state is associated with a particular execution, and each execution is delayed for a fixed time interval. Inconsistencies are identified by comparing the leading state (that optimistically processes game events without any additional delay) with the game states of the delayed executions that reorder and then process the received game events. If an inconsistency is detected, a rollback is performed by copying the (consistent) game state from the delayed execution to the leading execution and then the rolled back game events are re-processed in the proper order. Needless to say, a trade-off relationship exists between the number of game state copies maintained by each *GSS* and the number of game events that need to be re-processed in case of a rollback.

In [26, 27], *Mauve et al.* presented an optimistic approach for the consistency control in networked multiplayer games. This approach utilizes the *local lag* control mechanism combined with a modified Time Warp (executed only when necessary) in the attempt of solving the trade-off relationship existing between responsiveness and consistency. In particular, the devised synchronization approach is based on the idea of intentionally decreasing the responsiveness of the application in order to eliminate short-term inconsistencies. According to the local lag approach, game events are delayed for a certain amount of time before being executed. This additional delay permits the reordering of the received events in order to minimize inconsistencies. However, the presence of this delay is not always sufficient to prevent short-term inconsistencies since game events might still arrive late due to jitter or packet loss in the

network. To restore consistency, the proposed scheme resorts to the Time Warp algorithm.

Other approaches simply assume that delaying the game event processing activity for a predetermined amount of time may be sufficient to guarantee a uniform evolution of the game state at different nodes in the system without any need to resort to rollbacks. Following this idea, *Diot and Gautier* depicted in [10] a synchronization mechanism which embodies an optimistic version of the well known conservative bucket synchronization algorithm. Their approach assumes that there exists a processing deadline (which defines a *time bucket*) and that a correct evolution of the game requires that all game events are received prior to this deadline. In essence, the idea behind this scheme is that of ordering and processing game events at the end of the time bucket. If some game events are not received before the time bucket expiration, dead reckoning techniques are exploited to compensate events losses. A similar approach was presented also in [22], however, in both cases, the main drawback is that dead reckoning does not ensure the full consistency of the distributed game state. Short-term inconsistencies may still arise if the bucket size is set excessively small. On the other hand, using large buckets may induce a severe responsiveness degradation. Further, a complex problem is that of fitting the bucket size with the unstable condition of the Internet where large and variable jitter values may be experienced.

In [23], *Li et al.* presented a continuous consistency control mechanism for supporting networked multiplayer games. Similar to [27], this approach rests on the idea that game events should not be considered as discrete updates of the game state, but rather as continuous actions in the game world. In this context, the authors proposed a relaxed time-dependent consistency control scheme which gradually synchronize the nodes in the system and ensures that the discrepancy among game states stored at each node, even if existing, never exceeds a predefined threshold.

Finally, *Knutsson et al.* propose in [20] the use of coordinators to solve update conflicts that may occur in a peer-to-peer architecture. In particular, they split the game state management into different classes handled by different coordinators within a group of interested peers. However, these coordinators evaluate game situations based upon a subjective point of view, thus raising some concerns about fairness. Aiming at guaranteeing fault tolerance, the authors also propose the use of a primary-backup protocol to address fail-stop failures of coordinators.

3 System Model

This Section surveys the main concepts that are at the basis of an obsolescence-based game processing strategy deployed over a mirrored game server architecture. Interested readers may find a deeper analysis of these concepts in [14].

With the term *mirrored game server architecture* we refer to a game architecture that employs a constellation of replicated *Game State Servers* (*GSSs*) geographically dispersed over the Internet [8,30]. Each *GSS* locally maintains a replicated representation of the state of the game. According to this approach, each player connects to the *nearest GSS* and communicates in the same way as in a classic centralized client-server architecture. With each new action performed by players connected to a given *GSS*, the *GSS* collects the corresponding event, notifies it to other *GSSs* (in a peer-to-peer fashion), updates the game state and, finally, communicates the newly computed game state to its connected players.

In our model, we assume that generated game events represent discrete actions in the game world. Under this assumption, it is well known that game state consistency can be guaranteed based on the order according to which game events are processed, i.e., each *GSS* has to process game events according to a correct order so as to ensure game state consistency [17,25].

An important characteristic of MMOGs deployed over *GSSs* is the possibility of having several participants sharing the same game experience at the "same real-time". Unfortunately, real-time performance is often an illusive objective to achieve over a best-effort network. Several adaptation techniques have been proposed to manage the notion of time in MMOGs. In this context, we define a *Game Interaction Threshold* (*GIT*) that represents a temporal limit on the acceptable delivery delay of a game event. If delivery delays of game events surpass this limit, the system is not interactive. Therefore, in order to guarantee full satisfaction to players, the time elapsed from the generation of a game event to its presentation to final users should always be kept below the *GIT* value. As extensively discussed in [2,5,12,33], typical *GIT* values are in the range of 100-200 ms.

Denoting with $T_g^p(e_i)$ the generation time of a given event e_i at a given *GSS* p, and with $T_d^q(e_i)$ the delivery time of e_i measured at another *GSS* q, we introduce a metric that measures the *Game Time Difference*, $GTD_{p,q}(e_i)$, between the generation time of an event e_i and its delivery time to connected *GSSs*.

Definition 1 (Game Time Difference, GTD) *Given two GSSs p and q and an event e_i generated at*

p and sent to q, the Game Time Difference is defined as the difference between the event delivery time e_i at q and its event generation time at p: $GTD_{p,q}(e_i) = T_d^q(e_i) - T_g^p(e_i)$.

This value is an estimation of the interactivity degree provided by the system during the game activity, as it takes into account the most relevant portion of the delay: the communication latency between communicating *GSSs*. We claim that interactive game applications are well supported only if the *GTDs* of the generated game events are kept within the limit provided by the *GIT* at all *GSSs*.

To measure the *GTD* of delivered events, we assume a notion of global time among *GSSs*. This goal may be accomplished exploiting a variety of different approaches, such as synchronized physical clocks [7, 11, 18, 28] or technological synchronization devices (i.e. GPS).

Classic synchronization schemes in networked games guarantee that a given game event trace is processed according to the same order by all the *GSSs* [8, 10, 26]; this is sufficient to ensure game state consistency [14]. However, it is well known that the use of synchronization approaches that strive to reconstruct the same totally ordered event trace at all the *GSSs* may greatly affect the degree of responsiveness in large-scaled MMOGs.

From this standpoint, recent studies demonstrated that, by exploiting the semantics of the game, not all game events are required to be processed in the same order at all *GSSs* for maintaining game state consistency [14,32]. Hence, the following notion of correlation among game events may be introduced:

Property 1 (Correlation) *Two game events, e_i and e_j, are said to be correlated iff different orders of execution of the two game events lead to different game states.*

Examples of correlated game events typically involve events that could also be generated by different players and that act on the same game elements. In some sense, game events result to be correlated when there is a semantic relationship among them. As an example, we may consider two players who wish to simultaneously pick up the same object in a virtual world: to this aim, they generate two concurrent game events. Clearly, the order according to which the two game events are processed is important to determine who actually takes the object. Therefore, these game events must be processed at all *GSSs* based on a total (timestamp-based) order to not alter the game state. Two game events representing attempts to pick up different objects, instead, may be processed according to different orders at different *GSSs* without compromising the game state consistency; thereby, they are not correlated. Other examples

of non-correlated game events are represented by independent movements of different virtual characters (which typically represent the most commonly generated game events in a huge amount of distributed games).

Based on this notion of correlation, we introduce the following correlation-based order processing strategy:

Property 2 (Correlation-based Order) *Given two correlated game events, e_i and e_j, the two events are to be processed by all GSSs in the same order.*

Hence, according to Property 2, in order to provide players with a correct evolution of the game, it is sufficient that correlated game events are processed by all *GSSs* respecting their correct (timestamp-based) order. Conversely, no ordering guarantee is needed to process non-correlated events, as their delivery in different orders at different *GSSs* do not alter the consistency of the game state. In a few words, the processing at different *GSSs* of differently ordered event traces while respecting conditions of Property 2 still guarantees game state consistency. Such correlation-based order has the main advantage of reducing the synchronization overhead.

To introduce the notion of obsolescence, the following important consideration is in order: there exist many situations in games where fresher game events annul the importance of previous ones. For example, knowing the position of a character at a given time may no longer be important after a certain time period, if the position of the character changes. In other words, the newer movement event generated for that character makes *obsolete* the older one. We hence introduce the following notion of *obsolescence*:

Property 3 (Obsolescence) *Given two game events, e_i and e_j, generated at the same GSS p, we say that e_j makes e_i obsolete iff all the following conditions hold: i) $T_g^p(e_j) > T_g^p(e_i)$; ii) processing e_j without e_i leads to the same final state that would be reached if both events were processed in the correct order; iii) there do not exist other events e_k correlated to e_i which have been generated at any GSS within the time interval $[T_g^p(e_i), T_g^p(e_j)]$.*

It is evident that obsolescence and correlation have strict mutual relationships. In general, the presence of an event e_k correlated to e_i, and interleaved between e_i and e_j, may break the obsolescence relation between e_i and e_j. For instance, consider the game plot where the character *Alice* is moving through two subsequent moves: if *Bob* shoots at *Alice* during *Alice*'s motion, then no kind of obsolescence may be considered between the two subsequent *Alice*'s move-

ments. In such a case, in fact, the actual position of *Alice* is important to determine if the shot has either hit her or not.

Obviously, in order to check for correlation and obsolescence, some information has to be included in game messages exchanged among the nodes in the system. This information has to reflect the semantics of game events and may vary depending on the considered game. As a factual example, in [15] we provided a discussion on how to implement the notions of obsolescence and correlation within the context of distributed interactive storytelling systems. In particular, we showed that the use of an XML-based syntax to represent game events can ease the task of implementing relationships between characters (and their actions). Furthermore, in [6], another example has been provided on how to implement obsolescence and correlation in a car racing game.

Based on the notion of obsolescence, an *obsolescence-based processing* strategy may be devised.

Property 4 (Obsolescence-based Processing) *For each game event e_i and GSS p, one of the following conditions holds: either i) p processes e_i or, ii) p processes some further event e_j that has made e_i obsolete.*

Property 4 states that there is no need to process obsolete events to maintain the game state consistency, since it is guaranteed by the processing of fresher events.

A general notion of *game consistency* may be introduced which is as follows.

Property 5 (Game Consistency) *The game state, at any GSS, is said to be consistent iff the requirements of Property 2 and Property 4 are satisfied for each event trace.*

Meeting the conditions of Property 5 means that all *GSSs* agree on the same view of the game state as they deliver correlated events in the same order, while obsolete events may be safely discarded on a subjective basis.

In conclusion, satisfying the requirements of Property 5 entails that the final game state is not altered and an augmented interactivity is achieved by means of discarding obsolete events before processing them and permitting different processing orders for non-correlated events.

4 An Optimistic Obsolescence-based Synchronization Scheme

We present an *Optimistic Obsolescence-based Synchronization* (OOS) scheme that guarantees a correlation-based order processing for those events that are not discarded due to obsolescence.

Our proposed scheme satisfies the requirements of Property 5, thus guaranteeing the consistency of the game state at all *GSSs*.

In particular, OOS adopts an optimistic strategy inspired by the Time Warp algorithm [19]. Simply stated, as soon as an event e_i, coming from a *GSS* p, is received at any *GSS* q, a check is executed to detect whether e_i is obsolete. In the positive case, it is simply discarded. Otherwise, another check is executed to verify whether there exist other already processed events e_j (correlated to e_i) with a generation time larger than $T_g^p(e_i)$. For each event e_j meeting this condition, the system invokes a rollback procedure which is based on the standard incremental state saving technique [17]. It may be the case that some of these events e_j may become obsolete due to the reprocessing activity triggered by the rollback procedure. Needless to say, these events are dropped as soon as they are recognized as obsolete.

This scheme guarantees that game state consistency is preserved.

```
0 procedure OOS-receive-event-procedure() {
1    e_i := received event;
2    if (e_i is obsolete)
3       drop(e_i);
4    else {
5       Event_List := {e_j | e_j already processed ∧
                              e_j correlated to e_i ∧ T_g(e_j) > T_g^p(e_i)};
6       if (Event_List ≠ NULL) {
7          Rollback(Event_List);
8          Process(e_i);
9          for (each e_j ∈ Event_List)
10            if (e_j is obsolete)
11               drop(e_j);
12            else Process(e_j);
13       }
14       else Process(e_i);
15   }
16 }
```

Figure 1: Implementing OOS.

Our OOS algorithm is reported in Figure 1, which lists all the actions accomplished by a given *GSS* when a new game event is received. First, the *GSS* verifies if e_i may be already identified as obsolete (line 2). In this case, e_i is dropped (line 3). Otherwise, a check is carried out to control whether any game events e_j, correlated to e_i and generated after e_i, have already been processed (lines 5-6). If this check succeeds, then a rollback procedure is performed where all these events e_j are rolled back (line 7). At this point, e_i is processed (line 8), followed by the execution of all those rolled back events which are not obsolete (lines 9-12). Obsolete events are discarded during the rollback (lines 10-11). If the check fails, thus implying that no rollback

procedure is needed, then e_i is directly processed (line 14).

It is worth mentioning that our OOS scheme is devised to reduce the amount of rollbacks executed during the game processing activity w.r.t. Time Warp. Indeed, a rollback is performed only when receiving a late event that cannot be considered as obsolete. Instead, if obsolete, the event is dropped without invoking the rollback procedure. Moreover, a rollback is needed only if correlated events have been processed out of order and, even in this case, only non-obsolete events are reprocessed. In essence, our scheme has the positive effect of reducing the computational burden typically required to maintain the game state consistency.

It is also worth noticing that when an optimistic scheme is employed to synchronize *GSSs*, processed game events can be transmitted to players only when those events are no longer subject to possible rollback. In other words, the rollback strategy should not affect the game evolution as seen by players; otherwise, users could have the undesired perception of characters that, for example, jump from incorrect positions to correct ones, or come back to life when they have already been shown as killed.

In point of this fact, other proposals that refer to optimistic synchronization schemes for networked multiplayer games [9, 24] do not seem to take into consideration this problem. In essence, events are forwarded to players without any control on the stability of the rendered game state. Therefore, when a rollback occurs, the corresponding reprocessing is perceived by connected players as well.

We have a different position here. We claim that jerky renderings at player's side should be avoided even when an OOS strategy is employed. This may be achieved by notifying players only with a stable game state. While this approach slightly increases the amount of time spent by game events at the server side, the final degree of responsiveness can still be improved thanks to an obsolescence-based discarding mechanism as in our case. This claim is confirmed by results provided in Section 5.

We conclude this Section by mentioning that our OOS strategy has been implemented by exploiting a receiver-initiated communication protocol that utilizes NACKs (Negative ACKnowledgments) to provide the delivery guarantee only for non-obsolete events [13]. It is well known that TCP-based synchronization protocols have to be avoided at all costs while developing distributed games [29]. The motivation is that much of TCP's behavior, such as congestion control and retransmissions, is detrimental to meeting the MMOGs real-time constraint. Our approach, instead, exploits UDP to transmit events among *GSSs*. Moreover, since

we are concerned with the reliable delivery of non-obsolete events only, as soon as a receiving *GSS* detects that a non-obsolete event e_i is missing[1], a NACK is sent back to the sending *GSS*. Upon receiving a NACK, the sending *GSS* either retransmits e_i or transmits the most recently generated event e_j that has made e_i obsolete.

5 Experiments

5.1 Performance Metrics and Simulation

In this Section we report results obtained from the evaluation of our OOS mechanism. In particular, we measured: i) the provided interactivity degree, ii) the amount of dropped events, iii) the number of rollbacks, iv) the amount of reprocessed events within each single rollback.

To evaluate our scheme, we have simulated a general mirrored game server architecture comprising a variable number of *GSSs* (from five to eight). Without any loss of generality, we focused our attention on the event receiving aspect of one *GSS*, namely GSS_0, pretending that other *GSSs* are sending game events to it.

We simulated a best effort, non-reliable network (with a 10% message loss probability on average). Following the literature [5, 12], the transmission delay for each event was obtained based on a lognormal distribution whose parameters are reported in Table 1. In particular, values in Table 1 correspond to the average values and standard deviations of the network latencies that characterize the network links between each sending *GSS* and GSS_0. The average event size (200 Bytes) and the event generation rate at each *GSS*, are inspired by the online gaming literature as well [5, 12, 30]. Finally, the *GIT* is a fundamental parameter in our scheme and its value may be differently chosen depending on the considered game. Taking inspiration from [2, 5, 12], we set the *GIT* equal to 150 ms.

Table 1: Configuration of the *GSSs* (ms).

	GSS_1	GSS_2	GSS_3	GSS_4	GSS_5	GSS_6	GSS_7
Latency Avg	15	40	75	90	80	30	100
Latency Std Dev	10	15	30	10	20	15	25

Regarding the frequency according to which events were generated at a given *GSS*, we shaped the event generation by means of a lognormal distribution whose *Average Inter-Departing*

[1] based on sequence numbers

Time (AIDT) was set equal to 45 ms, with a standard deviation of 10 ms. This configuration represents a scenario of intense load traffic [5, 12, 30].

In our evaluation, we varied the number of simulated players from 25 to 90. These values can be justified as follows. It is realistic to suppose that a player is able to generate a new event every 200-500 ms (e.g., FPS games). This means that 2-5 game events per second are generated by each player. Since we set the AIDT equal to 45 ms at each *GSS*, we obtain a total number of player per *GSS* ranging from 5 to 11. Considering the total number of servers (from 5 to 8) we can approximate the final amount of players interacting at the same time as ranging from 25 to 90.

We conducted our experiments with event traces containing 1000 events for each sending *GSS*. We considered different event trace configurations, where the probability that an event is non-correlated to other events was set equal to 50%, 60%, 70%, 80%, 90%, respectively. Clearly, the higher the probability of non-correlation, the higher the number of events that will become obsolete during the game evolution. Indeed, a higher non-correlation probability entails more chances for an event e_j to make obsolete a preceding one e_i. Indeed, it is more likely that no events e_k have been generated by other players which break the obsolescence relation among e_j and e_i. Moreover, the higher the probability of non-correlation among game events, the lower the number of events that will be subject to rollback.

Where not differently stated, we have set the non-correlation probability equal to 90%; this represents a realistic scenario for a vast plethora of possible games. Indeed, we claim that this particular configuration is a good approximation of a real trace of game events as generated during a game session. For instance, in many first person shooter games, independent (i.e., non-correlated) movement actions performed by different characters largely surpass critical movement/shoot ones [1, 30].

As a confirmation of our claim, an extensive study of players' behavior on Quake 3 is presented in [3]. In that paper, *Armitage* reported a measure of the average number of kill actions per minute as a function of the median ping time between client and server; that measure helps us in providing a further rationale for considering 10% as the percentage of critical events in our simulations. In fact, in [3] it is shown that 80% of the players are located within 150-180ms of range from the server and experience a number of kill actions per minute, on average, comprised in the interval between 1.60 and 3.25. Considering the median of this interval (2.425) we obtain 0.04 kill events per second. With an AIDT of 45ms at client side, 22.22 game events

are generated every second. Therefore, the resulting percentage of kill events over the whole set of game actions amounts to just 0.18%. Pessimistically assuming that a kill event can be issued only after an average number of 50 correlated actions (e.g., various shots, movements of the character into the location where it would be shot or out of the position where it would have been shot) we get 9% of critical events. Therefore, 10% of correlated events and 90% of non-correlation (and obsolescence) probability represent a realistic configuration for online games simulations. Nevertheless, to provide a more general experimental scenario, we have evaluated configurations adopting various non-correlation probabilities.

Finally, in our experiments we contrasted five different synchronization schemes:

1. our OOS strategy;
2. the already mentioned COS-based Interactivity Restoring scheme described in Section 1 (hereafter denoted as COS-1) [13];
3. the already mentioned COS-based Interactivity Loss Avoidance scheme described in Section 1 (hereafter denoted as COS-2) [30];
4. a traditional optimistic synchronization algorithm, inspired by the Time Warp scheme (TW);
5. a traditional Conservative Synchronization (CS) approach that does not resort to any mechanism to restore interactivity nor utilizes any dropping strategy.

5.2 Results: Interactivity Degree

We intend to demonstrate here the benefits on the provided interactivity degree attainable by resorting to our OOS approach. To this aim, we compare the five aforementioned mechanisms by measuring the percentage of game events arrived at GSS_0 with a *GTD* value larger than *GIT*. To fully appreciate the results, we have to keep in mind that our experimental analysis also includes the computational cost paid for checking obsolescence and correlation, and to perform rollbacks when necessary. In particular, Figures from 2 to 5 report the average percentage of *GDTs* above *GIT*, depending on the number of involved *GSSs*, as a function of the non-correlation probability. According to all the experimental configurations, OOS outperforms the other schemes regardless of the number of sending *GSSs*. It is also worth noticing that CD and TW curves lie horizontal, as standard CD and TW schemes do not benefit from obsolescence and correlation.

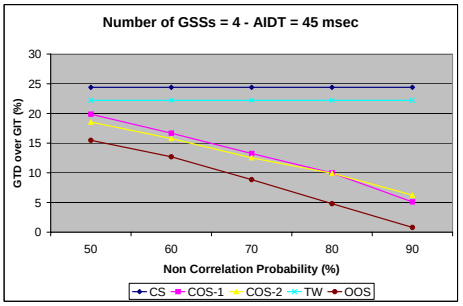

Figure 2: % Events with *GTD* over *GIT*; 4 *GSSs*.

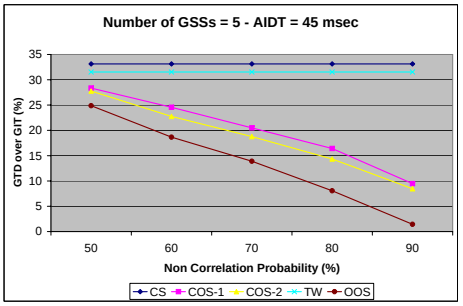

Figure 3: % Events with *GTD* over *GIT*; 5 *GSSs*.

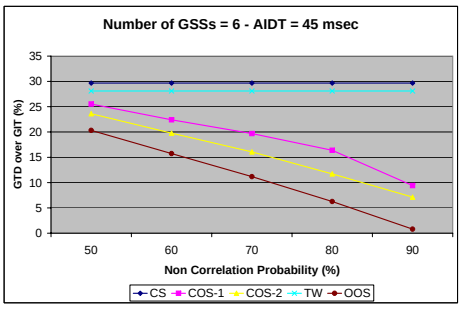

Figure 4: % Events with *GTD* over *GIT*; 6 *GSSs*.

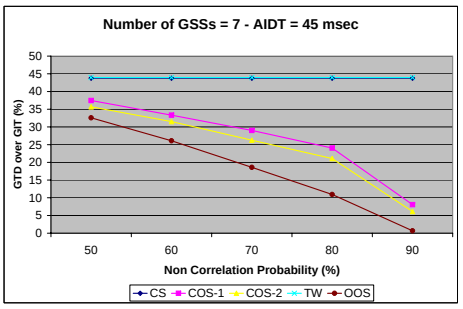

Figure 5: % Events with *GTD* over *GIT*; 7 *GSSs*.

Another proficient tool for evaluating the compared schemes is represented by the cumulative function of the *GTDs*. This value measures the probability of having a *GTD* lower than a specified value. In substance, the more the line reported in Figures from 6 to 9 is concentrated on the left side of the chart, the higher the interactivity degree provided by the system. Coherently with the previous outcomes, OOS outperforms all the other schemes, thus guaranteeing an augmented interactivity degree.

In essence, these results confirm that: i) our OOS strategy is better suited for games w.r.t. COS; ii) traditional optimistic approaches may gain great benefits from obsolescence and correlation notions.

5.3 Results: Amount of Dropped Events

An interesting measure is concerned with the amount of obsolete events that are dropped to report (and maintain) the interactivity degree within an acceptable value. Indeed, while these

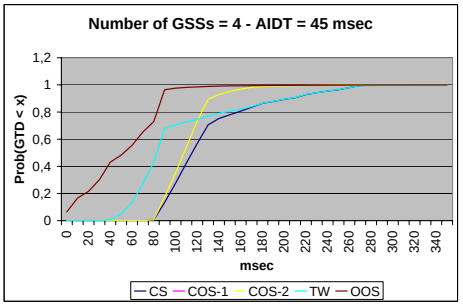

Figure 6: Cumulative Function of the *GTDs*; 4 *GSSs*.

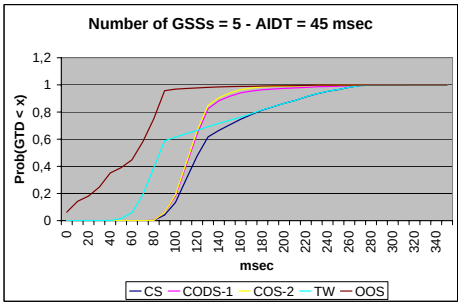

Figure 7: Cumulative Function of the *GTDs*; 5 *GSSs*.

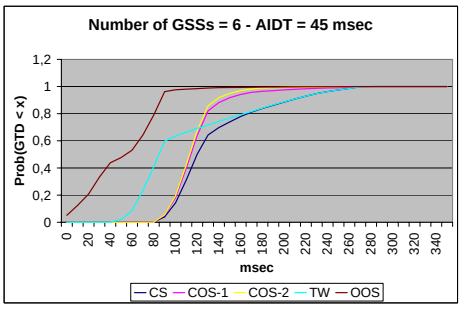

Figure 8: Cumulative Function of the *GTDs*; 6 *GSSs*.

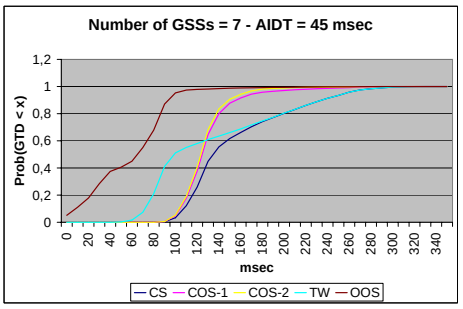

Figure 9: Cumulative Function of the *GTDs*; 7 *GSSs*.

dropped events are obsolete and do not influence the final game state, they are still part of the game visual evolution. Dropping too many obsolete events could result in sudden ”jumps” and jerky renderings that may affect the fluency of the game evolution. These unpredictable gaps in the game plot may annoy customers and should be avoided whenever possible.

Needless to say, only the schemes that are able to drop obsolete events are considered here. Figures from 10 to 13 report the percentage of dropped events using COS-1, COS-2 and OOS, espectively. Each Figure refers to a specific simulative scenario considering a different number of involved *GSSs*.

As shown in the charts, OOS reduces the number of discarded events in all the simulated configurations w.r.t. all the other schemes. This effect is derived from the fact that our OOS strategy accelerates fresher event processing, thus diminishing the possibility for an event to become obsolete.

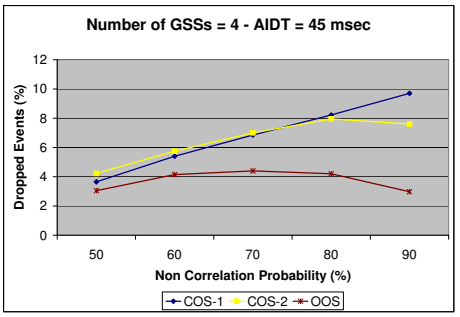

Figure 10: % Discarded Events; 4 *GSSs*.

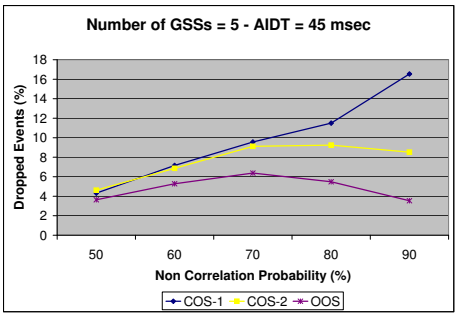

Figure 11: % Discarded Events; 5 *GSSs*.

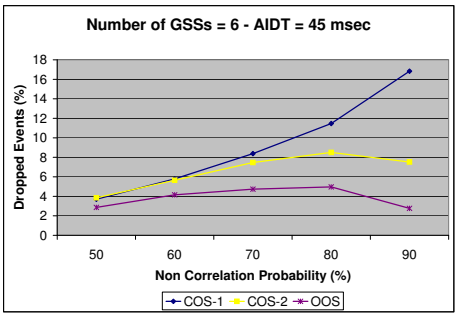

Figure 12: % Discarded Events; 6 *GSSs*.

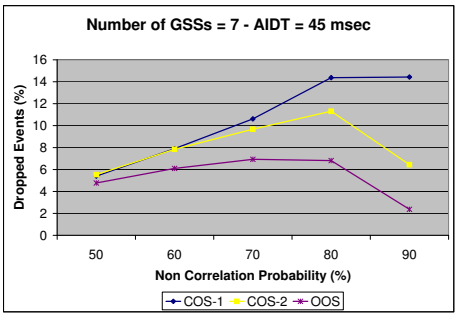

Figure 13: % Discarded Events; 7 *GSSs*.

5.4 Results: Rollback Ratio

We measured the number of rollbacks needed to maintain the consistency of the game state by comparing the two considered optimistic synchronization algorithms: TW and OOS. Since a reduced number of rollbacks implies a higher interactivity degree, this represents a fundamental metric.

In Figures from 14 to 17 we show the rollback ratio for TW and OOS. In simpler words, we measure the total number of rollbacks in the system over the total number of generated events for various non-correlation probabilities. Each chart refers to a different scenario with a different number of sending *GSSs*.

It is worth noticing that the TW curves lie horizontally, as standard TW does not gain benefits from obsolescence and correlation. Moreover, in all the considered configurations, OOS is able to outperform TW since it avoids to trigger the rollback procedure for obsolete

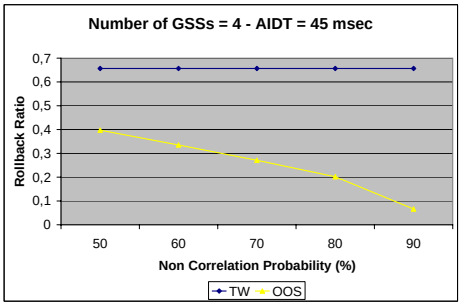

Figure 14: Rollback Ratio; 4 *GSSs*.

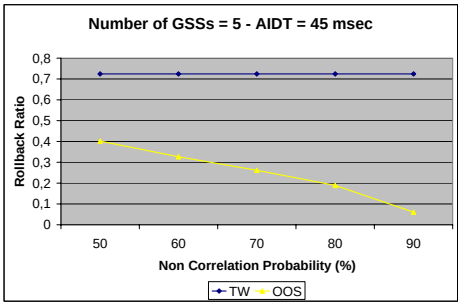

Figure 15: Rollback Ratio; 5 *GSSs*.

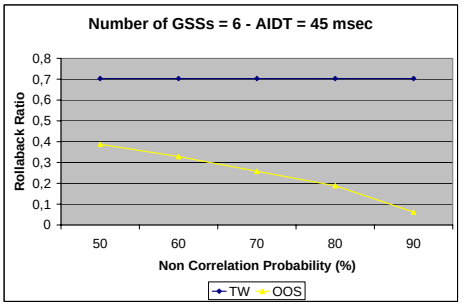

Figure 16: Rollback Ratio; 6 *GSSs*.

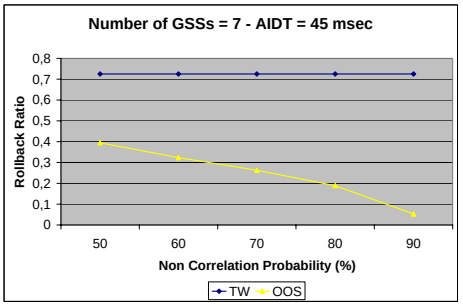

Figure 17: Rollback Ratio; 7 *GSSs*.

events.

Figures from 18 to 21 report the average number of re-processed events within a single rollback. In general, OOS reduces this value w.r.t. TW since it avoids the re-execution of those events that become obsolete during the evolution of the game (and during the rollback).

6 Conclusions

The volatile nature of the Internet can affect the usability of games when they are played by a huge number of customers dispersed over the network. The main cause of frustration for online players is represented by the fact that MMOGs generally have stringent interactivity requirements that are not fulfilled by the Internet's best-effort service model. In the attempt to alleviate this problem, we have proposed an optimistic obsolescence-based game event synchronization scheme devised to support the distributed evolution of games which are deployed over a constellation of mirrored game servers. Our synchronization mechanism exploits the

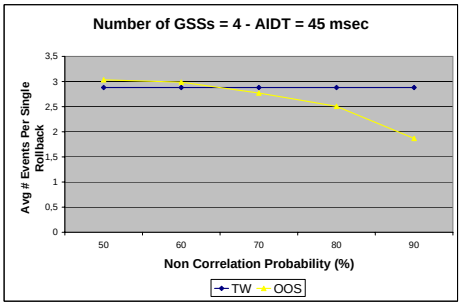

Figure 18: # Events per Rollback; 4 *GSSs*.

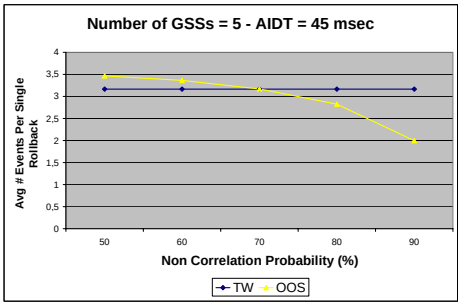

Figure 19: # Events per Rollback; 5 *GSSs*.

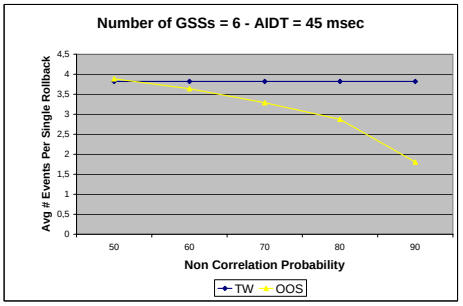

Figure 20: # Events per Rollback; 6 *GSSs*.

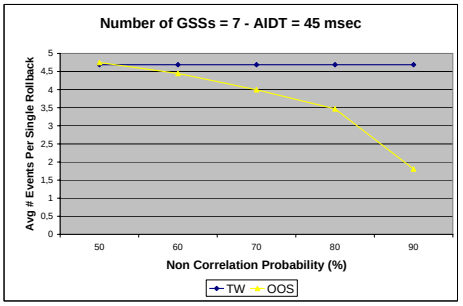

Figure 21: # Events per Rollback; 7 *GSSs*.

semantics of the game to reduce the processing overhead at each mirrored server, while maintaining the full consistency of the game state.

Our claim is that an optimistic synchronization approach, coupled with the notions of obsolescence and correlation, is well suited for MMOGs. On one side, in fact, an optimistic approach guarantees a constant rate of the game state advancement and, on the other side, obsolescence and correlation permit the reduction of the number of game events subject to rollback, thus minimizing the corresponding event processing overhead.

We have provided experimental results that confirm that an augmented interactivity may be guaranteed to players if our approach is adopted.

Acknowledgments

This work is partially supported by the Italian Ministry for University and Research (M.I.U.R.) via the InterLink project. Moreover, we feel indebted to the anonymous referees for their helpful review of this article.

References

[1] Half-life. http://counterstrike.sierra.com/.

[2] IEEE Std 1278.2-1995. IEEE Standard for Distributed Interactive Simulation - Communication Services and Profiles. Technical report, IEEE.

[3] J. Armitage. An Experimental Estimation of Latency Sensitivity in Multiplayer Quake 3. In *Proceedings of 11th IEEE Conference on Networks (ICON)*, September 2003.

[4] M. Billinghurst and H Kato. Collaborative augmentes reality. *Communications of the ACM*, 45(7):64–70, July 2002.

[5] M.S. Borella. Source models for network game traffic. *Computer Communications*, 23(4):403–410, February 2000.

[6] S. Cacciaguerra, S. Ferretti, M. Roccetti, and M. Roffilli. Car racing through the streets of the web: a high-speed 3d game over a fast synchronization service. In *Proceedings of International ACM World Wide Web 2005 Conference, Poster Track*, Chiba, Japan, May 2005.

[7] F. Cristian. Probabilistic clock synchronization. *Distributed Computing*, 3(3):146–158, 1989.

[8] E. Cronin, B. Filstrup, S. Jamin, and A.R. Kurc. An efficient synchronization mechanism for mirrored game architectures (extented version). *Multimedia Tools and Applications*, 23(1):7–30, May 2004.

[9] E. Cronin, B. Filstrup, A.R. Kurc, and S. Jamin. An efficient synchronization mechanism for mirrored game architectures. In *Proceedings of the 1st workshop on Network and system support for games*, pages 67–73. ACM Press, 2002.

[10] C. Diot and L. Gautier. A distributed architecture for multiplayer interactive applications on the internet. *IEEE Network Magazine*, 13(4), july/august 1999.

[11] R. Drummong and O. Babaoglu. Low-cost clock synchronization. *Distributed Computing*, 6(3):193–203, 1993.

[12] J. Farber. Network game traffic modelling. In *Proceedings of the 1st workshop on Network and system support for games*, pages 53–57. ACM Press, 2002.

[13] S. Ferretti and M. Roccetti. The design and performance of a receiver-initiated event delivery synchronization service for interactive multiplayer games. In S. Natkin Q. Mehdi, N. Gough, editor, *Proceedings of the 4th International Conference on Intelligent Games and Simulation (Game-On 2003)*, London, UK, November 2003.

[14] S. Ferretti and M. Roccetti. A novel obsolescence-based approach to event delivery synchronization in multiplayer games. *International Journal of Intelligent Games and Simulation*, 3(1):7–19, March/April 2004.

[15] S. Ferretti, M. Roccetti, and S. Cacciaguerra. On distributing interactive storytelling: Issues of event synchronization and a solution. In Springer-Verlag, editor, *2nd International Conference on Technologies for Digital Storytelling and Entertainment (TIDSE 2004), LNCS 3105*, pages 219–231, Darmstadt, Germany, June 2004.

[16] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.

[17] R.M. Fujimoto. *Parallel and Distribution Simulation Systems*. John Wiley & Sons, Inc., 1999.

[18] R. Gusella and S. Zatti. The accuracy of clock synchronization achieved by tempo in berkeley unix 4.3bsd. *IEEE Transactions of Software Engineering*, 15(7):47–53, July 1989.

[19] D.R. Jefferson. Virtual time. *ACM Transactions on Programming Languages and Systems*, 7(3):404–425, 1985.

[20] B. Knutsson, H. Lu, W. Xu, and B. Hopkins. Peer-to-peer support for massively multiplayer games. In *Proceedings of the Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2004)*, pages 96–107. IEEE, March 2004.

[21] J. Kumagai. Fighting in the streets. *IEEE Spectrum*, 38(2):68–71, February 2001.

[22] K.W. Lee, B.J. Ko, and S. Calo. Adaptive server selection for large scale interactive online games. In *Proceedings of the 14th international workshop on Network and operating systems support for digital audio and video*, pages 152–157. ACM Press, 2004.

[23] F.W.B. Li, L.W.F. Li, and R.W.H. Lau. Supporting continuous consistency in multiplayer online games. In *MULTIMEDIA '04: Proceedings of the 12th annual ACM international conference on Multimedia*, pages 388–391. ACM Press, 2004.

[24] M. Mauve. Consistency in replicated continuous interactive media. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, pages 181–190. ACM Press, 2000.

[25] M. Mauve. Distributed interactive media, 2000.

[26] M. Mauve, S. Fischer, and J. Widmer. A generic proxy system for networked computer games. In *Proceedings of the 1st workshop on Network and system support for games*, pages 25–28. ACM Press, 2002.

[27] M. Mauve, J. Vogel, V. Hilt, and W. Effelsberg. Local-lag and timewarp: Providing consistency for replicated continuous applications. *IEEE Transactions on Multimedia*, 6(1):47–57, February 2004.

[28] D.L. Mills. Internet time synchronization: the network time protocol. *IEEE Transactions on Communications*, 39(10):1482–1493, October 1991.

[29] M.M. Oliveira and T. Henderson. What online gamers really think of the internet? In *Proceedings of the 2nd workshop on Network and system support for games*, pages 185–193, 2003.

[30] C.E. Palazzi, S. Ferretti, S. Cacciaguerra, and M. Roccetti. On maintaining interactivity in event delivery synchronization for mirrored game architectures. In *Proceedings of the 1st IEEE International Workshop on Networking Issues in Multimedia Entertainment (NIME'04)*, volume GLOBECOM 2004 - Satellite Workshop, Dallas, USA, November 2004.

[31] C.E. Palazzi, S. Ferretti, S. Cacciaguerra, and M. Roccetti. A rio-like technique for interactivity loss avoidance in fast-paced multiplayer online games: a preliminary study. In M. Merabti, J. Schell, N. Lee, C. Lindley, and A. El Rhalibi, editors, *Proceedings of the 2nd Annual International Workshop in Computer Game Design and Technology (GDTW 2004)*, pages 113–119, Liverpool, UK, November 2004. ACM Press.

[32] C.E. Palazzi, S. Ferretti, S. Cacciaguerra, and M. Roccetti. Interactivity-loss avoidance in event delivery synchronization for mirrored game architectures. *IEEE Transactions on Multimedia*, October 2005. Accepted for publication.

[33] L. Pantel and L.C. Wolf. On the suitability of dead reckoning schemes for games. In *Proceedings of the 1st workshop on Network and system support for games*, pages 79–84. ACM Press, 2002.

[34] T.M. Rhyne. Computer games and scientific visualization. *Communications of the ACM*, 45(7):40–44, July 2002.

[35] N. Shachtman. New army soldiers: Game gamers. *Wired News*, October 2001.

[36] M. Tsang, G. Fitzmaurice, G. Kurtenbach, and A. Khan. Game-like navigation and resposiveness in non-game applications. *Communications of the ACM*, 46(7):56–61, June 2003.

[37] C. Wagner, M. Schill, and R. Manner. Intraocular surgery on a virtual eye. *Communications of the ACM*, 45(7):45–49, July 2002.