IMPROVING RESPONSIVENESS BY LOCALITY IN DISTRIBUTED VIRTUAL ENVIRONMENTS

F.Baiardi, L.Ricci
Dipartimento di Informatica,
L.go B.Pontecorvo 3, 56125 - PISA
{baiardi,ricci@di.unipi.it}

L.Genovali
IMT,Institutions, Market, Technologies,
Piazza S.Ponziano,6 - LUCCA
l.genovali@imtlucca.it

KEYWORDS

Network, multiplayer, latency, consistency

ABSTRACT

The recent diffusion of wide area networks distributed applications, like distributed virtual environments ($DVEs$), for instance massively multiplayer games, requires the definition of proper consistency models and protocols. Models like perceptive consistency take into account the interactivity of these applications by defining a set of real time constraints. This paper presents *MultiLags*, a protocol implementing perceptive consistency which exploits DVE locality to dynamically refine the time constraints according to the network conditions. An implementation of the protocol is presented. A set of experimental results show the effectiveness of the *MultiLags* approach.

INTRODUCTION

The diffusion of wide area networks has recently supported the development of distributed applications like chats, shared whiteboards, multi-player games and distributed interactive simulations. The large complexity of these applications requires the definition of suitable models and development environments. In this context, the definition of novel consistency models and protocols is a primary issue.

Consistency models for distributed systems have been first proposed within the parallel and distributed programming research area. Classical approaches, like Lamport's *Sequential* or *Causal Consistency* [6] have been widely adopted to model the behavior of concurrent shared memory applications. While some distributed applications can still exploit these approaches, their revision may be required when considering interactive applications with soft real time constraint, like distributed virtual environments. A distributed Virtual Environment, DVE, simulates a virtual world where a set of users located at geographically distributed hosts interact. Massively Multiplayer Online Games (MMOG), like World of Warcraft, and distributed military simulations are currently the most typical examples of DVE.

The definition of proper models for $DVEs$ is especially challenging, because it should take into account the notion of $time$ by describing new properties, like *simultaneity* of events generated at geographically distributed hosts and *instantaneousness* in the perception of events. On the other hand, properties like causality and concurrency of events must be always preserved.

Simultaneity guarantees that each host hosting the allocation of interest perceives at the same time any event generated within the DVE, independently of communication latencies. This is required to avoid fuzzy situations, for instance simultaneity of events may prevent a dead player from shooting [8]. As a matter of fact, suppose two players A, B, controlled by users located at remote hosts, shoot each other at the same instant of time. Since the shooting is simultaneous, the death of the players must be simultaneous as well. Instead, if the latency between A and B is too high, the death of B may be perceived by A before its shooting.

[3] presents a model integrating causality, simultaneity and instantaneousness and [7]defines a protocol implementing this model based on the notion of *local lag*. The basic idea is to delay the rendering of events generated by an host of an interval of time Δ, i.e. the local lag, in order to hide the delay due to network latencies in the notification of events to other hosts. If Δ is statically defined it must be overestimated because of latency jitter. This may decrease the responsiveness of the application and favor cheating.

This paper introduces *MultiLags*, an extension of the local-lag approach where the value of Δ may be dynamically determined. Furthermore, our approach exploits the locality characterizing DVE to associate multiple values of Δ to distinct groups of interacting users, for instance groups of players located at different regions of the DVE. This introduces several problems. First of all, a mechanism to detect these groups has to be defined. Then, a protocol to dynamically determine the value of Δ has to be defined as well. After describing the perceptive consistency model and the local-lag technique , we introduce the *MultiLags*. Finally we discusse the implementation problems of *MultiLags* and show some experimental results.

RELATED WORK

Perceptive Consistency [3] is a wall-clock time based model integrating simultaneity, instantaneousness and causal ordering. The model requires synchronized physical clocks for the hosts cooperating in the DVE application. Currently, this is feasible by exploiting the *Network Time Protocol* (NTP) distributed service [9] which guarantees an acceptable approximation, less than 50 ms, in clocks synchronization.

Let us consider a set S of events generated at different

Proceedings 21st European Conference on Modelling and Simulation
Ivan Zelinka, Zuzana Oplatková, Alessandra Orsoni ©ECMS 2007
ISBN 978-0-9553018-2-7 / ISBN 978-0-9553018-3-4 (CD)

hosts at the same wall-clock time. Perceptive consistency requires that

- $simultaneity$ any host perceives at the same time the set of events generated at different hosts
- any host perceives any event a limited amount of time Δ after its generation. The value Δ corresponds to the $Responsiveness$ of the application.

The value of Δ has to be carefully chosen by considering human sensing capabilities. A proper choice may guarantee both instantaneousness and simultaneity. Furthermore, casual ordering of events is guaranteed through physical clocks. Since an absolute clock synchronization is not feasible, Δ must include the clocks drift as well.

The main problem in guaranteeing *Perceptive Consistency* is due to the network latency. [7] defines a protocol implementing perceptive consistency and based upon the notion of *Local Lag*. Consider an event E generated by a user interaction, for instance a mine explosion, at host H at wall-clock time T. H delays the visualization of the event of Δ, the application responsiveness. Furthermore, it notifies to any other host that E has to be shown to the user at the instant of time T +Δ. The Δ delay hides network latency and guarantees, in absence of packet loss, simultaneity of events at different sites. In this way, Δ defines both the value of the responsiveness and that of the local delay, i.e. the $LocalLag$. In the following we will refer to responsiveness, rs.local-lag, according to the context.

In absence of packet loss and of network congestion, this simple protocol guarantees simultaneity and causal ordering of events. Obviously, Δ has a negative impact on instantaneousness, but if the value of response time is smaller than a given threshold, corresponding to human sensor capabilities, the delay is not perceived by the interacting users.

The *Local Lag* approach has two main drawbacks. First, the absence of packet loss and of network congestion cannot be guaranteed. This implies that a set of recovery mechanisms has to be defined, for instance backward recovery mechanisms like time-warp [7] or forward recovery ones, like rendez-vous [4]. The second problem concerns the definition of a proper value of the responsiveness. In [7] this value is defined before application starts and cannot be modified during its execution, according to the network conditions. Its definition depends upon the characteristics of both the application and the network. For instance, a latency of 1 ms may characterize a LAN, while one of 150-200 ms a wide area network. The definition of a proper value is critical, because low values increase the number of inconsistencies, while large ones cannot guarantee instantaneousness in the perception of the events and may favor cheating. As a matter of fact, if the events are received largely before their deadlines, the user may exploit the received information for its benefit.

In [5], a technique to dynamically modify the value of the responsiveness of an application is presented. According to this approach, each host dynamically probes the network conditions and modifies the value to reflect these conditions, i.e. responsiveness may be increased if the network is congestioned or it may be decreased when the network latencies are low. Since probing the network conditions can introduce a large overhead in the application, [5] proposes to collect the number of late messages with respect to their timestamps and the number of lost messages. This information can be collected by simply analysing the messages exchanged by the application. Since *heartbeat messages*, i.e. messages exchanged among the players to notify their position, are the events notified more frequently, they can be exploited to test network conditions. Packet loss can be simply computed by identifying the message through an unique sequence number. In the following the local information collected by a player P about the status of the network will be referred as local network view, LNV of P. [5] presents a set of simulations to test the effectiveness of the dynamic local lag approach, but no implementation in a realistic networked environment.

THE MULTILAGS APPROACH

This section introduces *MultiLags*[10], an approach exploiting the locality which characterizes $DVEs$ to improve consistency. We suppose that no central server coordinates the game state. Hence, the hosts notify each other game events according to a fully distributed computational pattern.

The approach proposed in [5] has been extended in order to exploit locality. As a matter of fact, each player of a multiplayer game mainly interacts with a subset of the other entities of the DVE, in general those located in its surroundings. This set depends on the player sensing capabilities, for instance its sight, its equipment and so on. This locality has been modeled through the notion of area of interest [1], [2] the region of the virtual world including entities each player can interact with. Two kinds of area of interest have been defined. Static areas of interest are defined by statically partitioning the shared world into regions. For instance a region may correspond to a room or to a city. A dynamic or mobile area of interest is the area of the virtual world surrounding each player and it changes when the player moves. Further areas of interest may be defined to model different events. Consider, for instance, the example in Fig. 1. Avatar B belongs to the area of interest of A. Now, suppose that the bomb represented by the black square generates an explosion which kills B. In this case A perceives the death of B, but it is not able to perceive the cause of its death. If the application requires that the causally ordered events are perceived by the player, the computation of the areas of interest should take into account not only the sensing capabilities of the player, but also the area of perception of any event.

Areas of interest can be exploited to improve the responsiveness in DVE to define distinct values of responsiveness for the players. The $MultiLags$ approach dynamically pairs a distinct responsiveness value with groups of players located in distinct regions of the virtual world. It is worth noticing that even if the players located in the same region of the virtual world may be

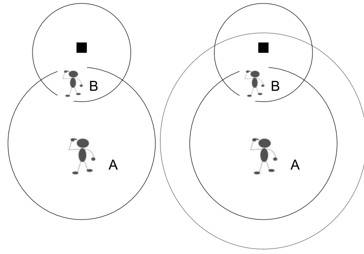

Fig. 1. Areas of Interest

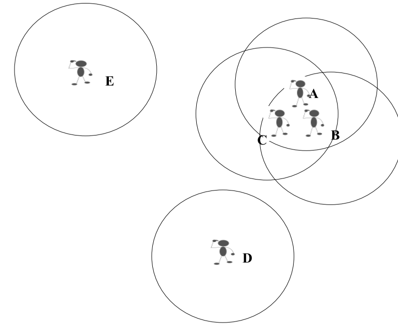

Fig. 2. Detecting Groups of Mutually Interacting Players

geographically distributed in the network, the goal of our approach is to optimize the computation of the responsiveness according to the condition of the network connections among the players. For instance, if all the players in the same region of the virtual world are located in the same country, the network delay may be smaller than the one of a group including players located in distinct countries.

It is worth noticing that, at any instant of time, all the players in an interaction group should have *the same value of responsiveness* in order to guarantee simultaneousness and a correct ordering of events. This requires a distributed consensus among the players, since any one may experience different network status. This is true even if just one pair of players is considered, because of different network routes and latencies between them, as shown in the following example.

Example 1: *Consider two interacting players A and B and suppose that the value of responsiveness is 100 ms. Suppose that A receives messages from B on average 50 ms. in advance with respect to their timestamps, while B receives messages from A on average 10 ms in advance with respect to their timestamps. If each host computes the new responsiveness value apart, A may reduce its response time, say to 60 ms, while B may leave the responsiveness unchanged. This will introduce inconsistencies on B, that will receive notification from A with an average delay of 40 ms.*

Computing the Local Network Views is therefore only the first step to implement the dynamic local lag technique. A further step is required to reach a distributed consensus on the new responsiveness value which is obtained by considering the LNV of the all interacting players. This requires that each player broadcasts its LNV to each other interacting player. This information can be transmitted through $heartbeat$ messages, which are periodically exchanged among the players in the same area of interest. Furthermore, each host periodically executes a procedure to combine its LNV with that of the other players in order to define the new responsiveness value. All the players execute this procedure roughly at the same time, and exploit the same operator to compute the responsiveness values. Finally, any host updates its responsiveness value.

When considering static areas of interest, each area of the DVE may be characterized through a distinct responsiveness value, because any player only interacts with other players in the same area of interest. Responsiveness may be computed by considering the network connections among the players belonging to that area. Each player dynamically updates this value when it moves from a region to another one.

The main problem of this solution is due to the dynamic acquisition of the new responsiveness value, when a player P moves into a new region. As a matter of fact, the new value may be computed only after some interactions with other players located in that region. In the simplest solution P exploits a predefined conservative value of responsiveness, but this will introduce inconsistencies because this value is not equal to that of all other players already located in that area. On the other hand, if the responsiveness is updated frequently, the number of the resulting inconsistencies is low. This solution may be refined by storing the responsiveness value of a region when leaving an area and by exploiting this value when re-entering that region. The predefined value may be exploited when visiting an area for the first time. This may be useful when player mobility is low or when a player re-enters an area after a small amount of time.

Responsiveness Prefetching is a more refined approach where each player starts accepting events from the players in a region R when approaching the border of R. This information can be exploited to prefetch both the updated responsiveness value and the right position of the players in R.

The application of the $MultiLags$ approach to mobile areas of interest is more complex because now it is more complex to detect groups of mutually interacting players, with the same responsiveness value. This is shown in the following example.

Example 2: *Consider the set of players S=$\{A,B,C\}$ in Fig. 2. They may reach a consensus about the value of responsiveness because any of them interact with any other one in S. This value may be different, for instance, from that of players D or E, located in different regions of the DVE. Let us now suppose that D gets in touch with B, but not with A and C, as shown in Figure 3. B may decrease its responsiveness value because of the low*

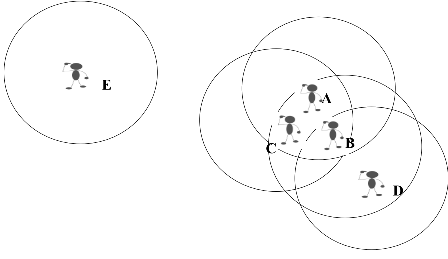

Fig. 3. Detecting Groups of Mutually Interacting Players

latency connection with D. Suppose also that an high latency connection exists between B and rs. A and C. This implies that A and C will experience a delay in any message received from B. This increases the number of inconsistencies generated by the application.

Hence, the definition of $MultiLags$ is more complex when mobile areas of interest are exploited. A first solution dynamically detects groups of interacting players. This corresponds to consider a *belongs to* relation among players such that A *belongs to* B iff A belongs to the area of interest of B. A group of mutually interacting players may be defined as a set S of players such that the *belongs to* relation defined for S is equal to its transitive and symmetric closure. While this condition is verified by all the players belonging to the same region when static areas of interest are considered, more complex protocols have to be adopted when dynamic areas of interest are exploited. For instance, each player may include in each notification a reference to any player interacting with him. Each player detects if the set of players interacting with it is equal to that received from other players. In this case, a distributed consensus among the players about responsiveness may be defined. Otherwise, each player may exploit a predefined value. As shown in the following section, our implementation of $MultiLags$ exploits static areas only, because the complexity of the other techniques is too large.

IMPLEMENTING MULTILAGS

This section describes an implementation of $MultiLags$ for $DVEs$ exploiting static areas of interest.

The communications among players belonging to the same area are implemented through a set of multicast groups, one for each area. Each player subscribes the multicast group G corresponding to a region R, when it enters R. While it moves within R, it exploits G to exchange event notifications with other players in R.

To delay a local event E of Δ, i.e. the value of the local lag, each player associates a timestamp with any event E and stores E in a data structure, in order to render E only after the local-lag has expired, when E will be inserted into the first rendered frame. Each player periodically executes the rendering of the scene. The frequency of rendering is determined by the *frame rate* of the host. An upper bound of 25 frame per second may be defined to constrain the frame rate, since an higher rate cannot be perceived by human senses and it would waste computational resource only. This corresponds to introduce an upper bound on the size of the data structure that records the delayed events as well.

Each event with the corresponding timestamp is sent to the multicast group as well. Furthermore, each player associates its LNV with each heartbeat sent to the multicast group. The pseudo code of the send procedure is the following:

Let E be an event generated at wall-clock time T
msg.timestamp = T + Δ
msg.LNV = LNV ;
msg.ID = MY_{ID};
msg.event = E;
send (msg)

The pseudo code describing the operations executed on a message reception is the following one

receive (msg);
T=msg.timestamp;
id=msg.ID;
store msg.LNV in a local structure at position id
MsgDrift=wall_clock - T;
LNV:= f (LNV,MsgDrift) ;

The host extracts the identifier of the host H which sent the message, the timestamp T and the LNV sent by the host H. Then, it records the LNV of H and exploits the drift between the timestamp of the message and the wall-clock time of the reception to update its LNV. To compute LNV different operators may be exploited. For instance the average temporal drift may be computed.

Each host periodically executes a procedure, *Update-Responsiveness* to refine the responsiveness value according to its LNV and to those received from other players. It is worth noticing that, while incrementing the value of responsiveness is always safe, a decrement may result in inconsistencies in the causal ordering of events.

Example 3: *Let us suppose that a player generates an event A at $t1$. A causes event B, which is generated at time $t2$. If the value of responsiveness is decreased after the occurrence of A, but before that of B, it is possible that B is perceived by the user before than A, thus violating the causality.*

Inconsistencies may be avoided by storing the timestamp T of the last event generated at each host. After the application of the *Update-Responsiveness* procedure, if any event gets a timestamp $T1$ lower that T, $T1$ is incremented in order to obtain a value greater than T. The value of the increment may be equal to the drift between the old responsiveness value and the new one. Notice that no inconsistencies can be generated by two causally ordered related events generated at two different hosts because the decrement of the response time is one order of magnitude smaller than that of network latency.

Fig. 4 shows the pseudo code of *Update-Responsiveness*. The interval of time between two consecutive executions of the procedure is denoted by π. The

```
if (Timer ≥ π) and (My_entry_time ≤ Wall_Clock - π ) {
    Hosts = { H : H_entry_time ≤ Wall_Clock - π }
    Hid = { id : id identifies H in Hosts }
    if (Hid ≠ 0){
        Max_ID = max { id ∈ Hid } ∪ My_ID
        if (My_ID == Max_ID)
            { LNV_Others=
                {LNV:LNV is Local Network View of host id,id ∈ Hid}
            NetDrift := f(LNV, LNV_Others)
            if NetDrift ≥ Threshold_a
                { Responsiveness = Responsiveness + Γ }
            else if NetDrift < Threshold_b
                { Responsiveness = Responsiveness - Γ }
            send Responsiveness to any host in my region
            Timer = 0;
            reset data strucutes} }
```

Fig. 4. Responsiveness Update

value of the timer is set to 0 after each execution of the procedure and the procedure is executed when this value is greater that π. Note that a fully distributed procedure may introduce inconsistencies in the computation of the refined responsiveness value because different hosts can compute different LNV, due to packet loss. For this reason, our solution is based on the dynamic selection of a host coordinating the computation of the refined responsiveness value for all hosts located in that area. Since each host is uniquely identified, for instance by its IP address, the host with the highest identifier may be dynamically selected as the coordinator. According to a $SPMD$ programming paradigm, each host executes an instance of *Update-Responsiveness*, but a single host, i.e. the coordinator, computes the new responsiveness value. Then, the updated value is broadcasted to any other host through the multicast group. The coordinator is chosen among the hosts which have already participated at one instance of the execution of *Update-Responsiveness*. In this way, hosts which have been in the region for a period of time shorter than π are not selected because the value of their LNV is not reliable, as they may have exploited an obsolete value to initialize their responsiveness. For the same reason, the LNV of the hosts which have entered the region after the last execution of *Update-Responsiveness* are considered unreliable and are not exploited to refine responsiveness. The network state, i.e. $NetDrift$ is computed by combining the network views of the different hosts by an operator, f, which must be defined according to the consistency degree we want to obtain. By considering the maximum value of the LNV we minimize the number of inconsistencies, but this corresponds to a low degree of responsiveness. Generally the choice of the operator is a trade off among responsiveness and consistency. If the resulting $NetDrift$ is higher or lower than two predefined threshold values, the responsiveness id increased rs. decreased of a predefined value Γ. In any case, un upper bound for this value is statically defined.

EXPERIMENTAL RESULTS

We have implemented *MultiLags* on 16 Athlon 2600+ hosts, connected by a Tricom Suoer Stack local network. To simulate wide area network conditions, we have introduced network delays generated according to an exponential distribution. This models both network congestion and jitter. We have considered a DVE partitioned into four rectangular regions, where a set of player moves. A distinct average latency has been associated to each region. The values associated to the region from 0 to 3 are, respectively, 80,120,160,200 ms. The *MultiLags* algorithm is executed every 5 seconds and, at each execution, the response time is incremented/decremented by 10 msec. The initial value of the response time is 100 msec.

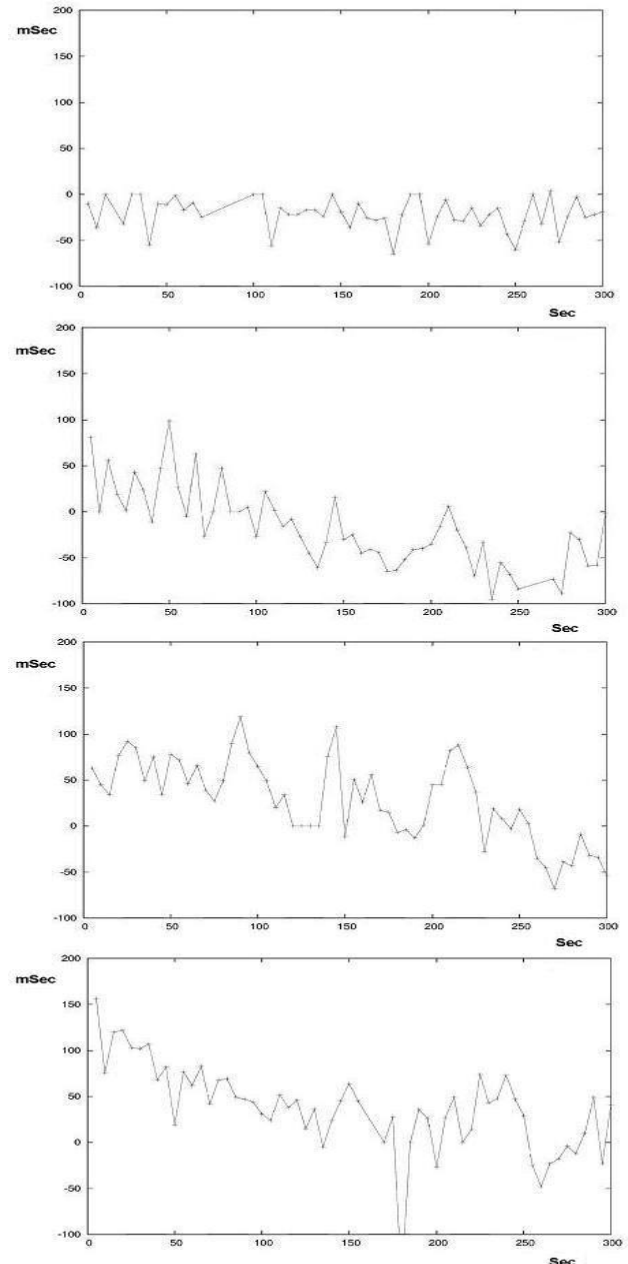

Fig. 5. Average Temporal Drifts in Different Regions

In Fig. 5 we consider the average drift between the message timestamp and the wall-clock at its reception. The four curves show the drifts of the four regions. The delay increases from the top curve, region 0, to the bottom

one, region 3. For each point of the x-axis corresponding to time t of the application execution, the y-axis shows the average temporal drift on the different hosts. We can notice that most message are received in time in region 0, where the value of latency is low. In other regions, a large number of messages are not received in time when the application starts. The figure shows that *MultiLags* is able to reduce the number of delays. As a matter of fact, after 250 msec, each message is received, on average, ahead of its deadline. This proves the effectiveness of our approach.

Fig. 6 shows the number of *Severe Errors*. A severe error is signaled when the drift between the timestamp and the wall clock at its reception is larger than 50 ms. We notice that while about no errors are detected when latency is low, their number increases with the latency.

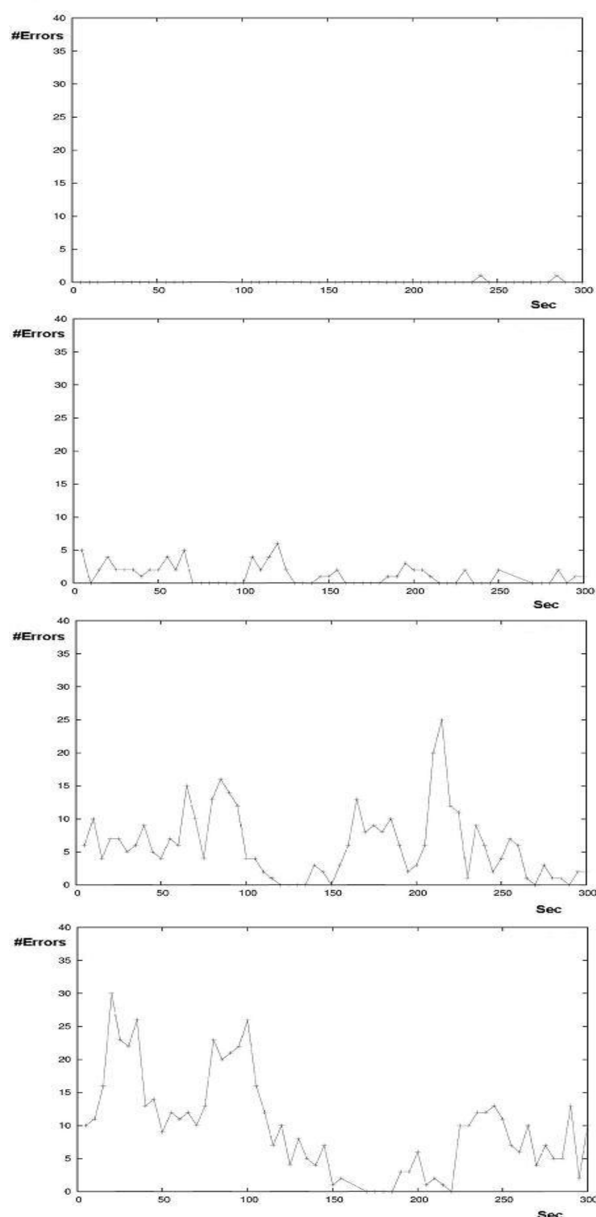

Fig. 6. Severe Errors

CONCLUSIONS

In this paper we have presented *MultiLags*, a protocol to implement perceptive consistency in $DVEs$. Our approach can dynamically modify the local-lag and associate different values of the local-lag to distinct regions of the DVE. We plan to refine our approach, by considering prefetching of responsiveness values by player approaching a new region. Furthermore, we plan to experiment our approach on a wide area network.

REFERENCES

[1] A.Bonotti, L.Genovali, L.Ricci DIVES: a Distributed Support for Networked Virtual Environments *Proceedings 20-th IEEE Conference on Advanced Information Networking and Applications AINA 2006*, Wien, April 2006.

[2] F.Baiardi, A. Bonotti, L.Genovali, L.Ricci. A publish subscribe support for networked multiplayer games Internet and Multimedia Systems and Applications (EuroIMSA 2007) Chamonix, France - March 1416, 2007, IASTED Press.

[3] Nicolas Bouillot, Eric Gressier-Soudan, Consistency models for distributed interactive multimedia applications, *ACM SIGOPS Operating Systems Review Archive* Volume 38, Issue 4, October 2004, Pages: 20-32.

[4] Angie Chandler, Joe Finney, Rendezvous: An Alternative Approach to Conflict Resolution for Real time Multi-user Applications, *13th Euromicro Conference on Parallel, Distributed and Network-based Processing* Lugano, Switzerland, February 2005

[5] Euisuk Hong, Dongman Lee, Eunkwang Park and Kyungran Kang , An Efficient Synchronization Mechanism Adapting to Dynamic Network State for Networked Virtual Environments, *Proceedings of SPIE Multimedia Computing and Networking 2003*, Ragunathan Rajkumar Editor, January 2003, pp. 24-33.

[6] Leslie Lamport, Time, Clocks, and the Ordering of events in a distributed system, *Communications of the ACM*, Volume 21 , Issue 7, July 1978, Pages 558-565, 1978.

[7] Martin Mauve, Jiirgen Vogel, Volker Hilt and Wolfgang Effelsberg, Local Lag and Timewarp: Providing Consistency for Replicated Continous Applications, *IEEE Transactions on Mutimedia*, February 2004, pages 47- 57 Volume: 6,1

[8] Martin Mauve, How to Keep a Dead Man from Shooting, *Proceedings of the 7th International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services*, Pages: 199-204, 2000.

[9] David L. Mills, Network Time Protocol (Version 3) Specification, Implementation and Analysis, *Request for Comments: 1305*, RFC 1305, March 1992 University of Delaware.

[10] Luca Genovali DiVES: un Sistema per la Gestione della Consistenza in Applicazioni Distribuite Real Time. *Master Thesis, Department of Computer Science, University of Pisa*, June 2005.

Laura Ricci is an assistant professor at the Department of Computer Science, University of Pisa. Her main interests are parallel programming, distributed systems, P2P systems. Her e-mail address is ricci@di.unipi.it and her Web page can be found at http://www.di.unipi.it/ ricci/

Luca Genovali is a Phd Student at IMT,Institutions,Marketing, Technologies, in Lucca. He is currently working on his Phd thesis 'Design and Evaluation of overlay topologies and consistency models for the development of P2P Distributed Virtual Environments'. His e-mail address is l.genovali@imtlucca.it