Fast Event Ordering and Perceptive Consistency in Time Sensitive Distributed Multiplayer Games

Nicolas Bouillot
CEDRIC-CNAM 292 rue St Martin
75141 Paris cedex 03 France
Email: bouillot@cnam.fr

KEYWORDS

Fully distributed games, consistency, dead reckoning, ordering of event, network latency, temporal user requirements, algorithm.

ABSTRACT

Distributed games provide to users an immersive and co-present virtual environment. In time sensitive games, this is achieved if remote perceptions of events respect some physical properties as the passing of time, the spatial positions, the dimensions... and if local updates are displayed instantaneously (fast responsiveness). As shown in the literature, spatio-temporal requirements are satisfied by dead reckoning coupled with timed consistency management. In this paper, we provide a consistency model for fully distributed game systems and an optimistic consistency management that works *friendly with* (or *without*) dead reckoning schemes. Our protocol presents two benefits: it provides a consistent state as fast as the network allows and it is well suited for both discrete and continuous media.

INTRODUCTION

Time-Sensitive Distributed Multiplayer Games are subject to consistency at the user level. In fact, each user interacts directly with its own "replica" which maintains the local game state and send local updates to the others. As these replica reflect the same virtual environment they are supposed to respect the temporal characteristics of the game. Fully distributed and mirrored games architectures deal with this kind of architecture (Cronin *et al.*, 2002). In this paper we are interested by a timed consistency (i.e. with strict temporal requirement) for games that provides highly interactive collaborations as in First Person Shooter games (FPS), in racing games or in fast action games. In a previous work (Bouillot & Gressier-Soudan, 2004), we have specified these requirements as follows:

1 For the usability of the game, each local updates (or actions) are played out instantaneously (i.e. as fast as the local computation can do for a fast responsiveness)
2 Each update occurs in the same order for each replica (free of conflicts)
3 The physical time between the playout of two updates is the same for all users. It is called *simultaneity* for updates coming from different sources and Δ legality for updates from the same source (spatio-temporal requirements)

In time sensitive distributed games, these requirements must be respected by the system. Let's look at an example, a racing game with two players:

- When a user move the car with its own device, the car must move sufficiently fast at the screen (item 1)
- When two remote cars (but close in the race) cross the finish line, the same car must be declared as the winner for the two users (item 2).
- Remote collisions are consensually detected if for the users, both cars are considered at the same place at the same time (*simultaneity*)
- The speed of a car is respected if the time between two successive positions of a given car is the same for the two users (Δ *legality*)

In Time-Sensitive Distributed Multiplayer Games, Dead Reckoning is used to achieve the first requirement

but also to reduce the amount of bandwidth used to send updates among players. In fact, for a predictable behavior, it is not necessary to send each state as the next states can be efficiently predicted (if the update message contains information that permits to predict future). For example, updates in a position stream can be optimized by sending absolute positions, velocity and direction (Aggarwal *et al.*, 2004). However, some consistency errors remain because the prediction is optimistic and users actions are sometimes unpredictable.

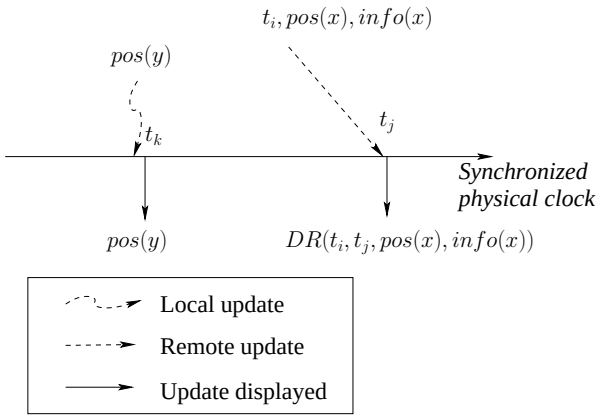

Fig. 1. *Dead Reckoning and the playout of the updates*

The figure 1 highlights how Dead Reckoning can hide the network latency to the user (according to the (Aggarwal *et al.*, 2004) method) while minimizing the update number. According to the item 1, local and remote updates must be immediately computed. Local updates (like $pos(y)$) are displayed as fast as possible, because the games must keep a good responsiveness. In other words the update and its playout occur approximately at the same physical time t_k (In fact the time taken by the local peripherals to play out the update). For the same reason, remote updates are computed to be displayed immediately when received.
However according to item 3, the received updates may have been computed at t_i (i.e. at the effective time of the action corresponding to the update). Then, the displayed update (at t_j) is the estimated position from the Dead Reckoning information $info(x)$ if x was at the position $pos(x)$ at time t_i. In this way dead reckoning allows the game to respect the previous requirements but errors still occur as each remote update locally played out are predicted. These errors can be repaired with new arriving real update, causing some divergent/convergent/divergent... effects (Pantel & Wolf, 2002).

Errors are tolerated (or accepted) if updates are not critical, as basic movements of a player, but it leads to different states if the update leads to an important decision (as collision detection for example) during the divergence phase. The Trailing State Synchronization (Cronin *et al.*, 2002) provides a solution to keep the consistency strong consistency with dead reckoning (and then the instantaneity of the local updates). It maintains a set of background states called the Trailing states where updates are delayed with a specified delay (the remote game replicas share a synchronized physical clock with NTP (Mills, 1989)). Then, when one of the delayed states allows to detect an inconsistency with the displayed state, the new consistent event is displayed and the old one discarded (for example with a rollback).

Protocols from (Mauve *et al.*, 2004; Bouillot, 2003; Cronin *et al.*, 2002; Gautier & Diot, 1998; Akyildiz & Yen, 1996) provide consistency management that maintains a consistent state. They can be used as Trailing States because in these protocols, local events are delayed to satisfy the spatio-temporal requirements. However, TSS (Cronin *et al.*, 2002), as (Mauve *et al.*, 2004; Gautier & Diot, 1998) use a global physical clock to decide *when* the updates must take effect in the Trailing states. It is function of time of the write ($T(c(x)v) = d + T(w(x)v$ where T is the time given by a (NTP-)synchronized clock the operation performed on the media instance x which returns the value v. w is the write operation and c is the handle operation, next section). They define d as a constant for all participants (and for a given Trailing State). TSS uses two trailing states, $d = 200ms$ for the first and $d = 400ms$ for the second. This allows to repair inconsistencies after 200ms or 400ms. This "static" provisioning of d can be improved if the consistency management take into account the network latencies. In fact, if d is chosen as a constant then a latency between users bigger than d would cause inconsistencies at each update. On the other side, if the latency is lesser than d, the state may be updated faster.

In this paper we propose an consistency management scheme adapted from the one proposed in (Bouillot, 2003) for Networked Musical Performance. We take into account the latency between players and show that it provide an ordering of updates that avoid potential conflicts (item 2). Thanks to our consistency management, the distributed game system take decisions as fast as possible, i.e. as fast as the network latencies allows.

A MODEL FOR FULLY DISTRIBUTED MULTIPLAYER GAMES

A distributed multiplayer game system is composed of a finite set of sequential processes $P = (p_1, ..., p_n)$ that interact with a set $X = (x_1, ..., x_m)$ of shared objects (called media instances). We denote these processes *replicas*, because they are supposed to maintain a consistent state of the virtual environment by exchanging media updates.

In our model, we identify the set of updates related to a particular object as a *media instance* (i.e. a stream). Thus a single *media instance* is composed of logical data unit (LDU's) (Steinmetz, 1996) resulting from *write* operation and that will be *handled* by the receiver's system. For example, a sample is a LDU in an audio stream and an action performed by an avatar (or a player) is a LDU in a stream of events. Continuous media have a constant or variable rate whereas discrete media have not but they share the same temporal requirements, i.e. simultaneity and Δ legality. Thus our model supports both kinds of media.

Intuitively, at the user's replica a *write* on a data corresponds to the generation of a new value for a media instance and a *handle* (associated to a write by its value) is the operation that decides when the write takes effect (the playout). A write of a value v into object x by process p_i is denoted $w_i(x)v$ and each value v written is considered as unique. In the same conditions a catch operation (the *handle* that returns the value coming from the previous write) is denoted $c_i(x)v$. In order to simplify the notation, op will denote either a write or a handle.

Previously introduced by (Raynal & Schiper, 1996) for Distributed Shared Memory, the local history $\widehat{h_i}$ of p_i is the sequence of operations issued by p_i. If $op1$ and $op2$ are issued by p_i and $op1$ is issued first, then we say $op1$ precedes $op2$ in p_i's process-order, which is noted $op1 \rightarrow_i op2$. Let h_i denotes the set of operations executed by p_i, then the local history $\widehat{h_i}$ is the total order $(h_i, \rightarrow_i)$.

A history $\widehat{H}$ of a distributed multiplayer game is a partial order $\widehat{H} = (H, \rightarrow_H)$ such as:

- $H = \bigcup_i h_i$
- $op1 \rightarrow_H op2$ if :
  i) $\exists p_i : op1 \rightarrow_i op2$ (in that case, $\rightarrow_H$ is called *process-order* relation)
  or ii) $op1 = w_i(x)v \wedge op2 = c_j(x)v$ (in that case, $\rightarrow_H$ is called *handled-from* relation)
  or iii) $\exists op3 : op1 \rightarrow_H op3$ and $op3 \rightarrow_H op2$

Each replica maintains a local physical clock that dates local events, i.e. local *writes* and local *handles*. More formally, we denote t_i the physical local clock of the process p_i. Then $t_i(w_i(x)v)$ dates locally and physically the write of the value v on object x. We notice that $t_i(w_j(x)v)$ is a non-sense as t_i can date only events that are local to p_i. Although we use only local clock to specify our requirements, we use T the universal clock to understand some global properties of the consistency model (as the delays between replicas).

THE PERCEPTIVE CONSISTENCY MODEL

In this section, we provide formal and intuitive definitions of the consistency requirements. We discard causality as it was shown in (Zhou *et al.*, 2002) that Lamport's causality is not adapted to interactive multimedia communications. In fact, causality is not adapted because it supposes that the virtualized environment is centralized (the processor) while in many time sensitive distributed games users' accesses of the virtual environment is not turn based but naturally parallel.

Concurrency and Conflicts

In this paper, we define that two operations $op1$ and $op2$ are concurrent if they are conflicting, more formally if $op1$ and $op2$ operate on the same object and $\exists i, j : c_i1 \rightarrow_i c_i2$ and $c_j2 \rightarrow_j c_j1$. For example if we have both $c_i(x)v \rightarrow_i c_i(x)u$ and $c_j(x)u \rightarrow_j c_j(x)v$. More intuitively, conflicts are caused in distributed multiplayer games by the fact that different replicas schedules handle operations in a different order. Intuitively our definition expresses the fact that updates are all seen in the same order by all the users.

Δ Legality

Inside the distributed multiplayer games and on the same media instance, the temporal relation between two local handles with their respective write (local or not) must be kept. For example for continuous media or for discrete media that reflect continuous event, such as movements where the speed must be maintained. Suppose that in figure 2 $w_2(x)v$, $w_2(x)u$ and $w_2(x)y$ are successive positions of the car driven by user p_2, then the speed of the car is maintained for p_1 if $d_1 = d_1^{'}$ and $d_2 = d_2^{'}$.

This relation is used in many streaming engines witch maintain a constant latency between writes and handles.

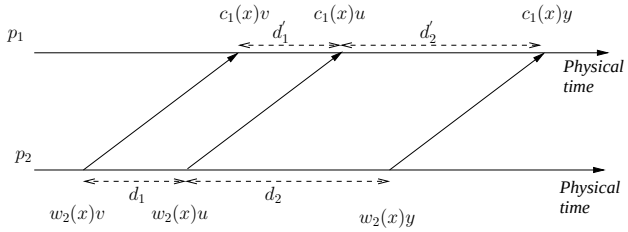

Fig. 2. *The Δ legality criterion*

We call this criterion Δ legality. More formally, an history $\widehat{H}$ is Δ **legal** if:
i) $\forall x \forall u, v \forall i, j : t_i(w_i(x)v) - t_i(w_i(x)u) = t_j(c_j(x)v) - t_j(c_j(x)u)$

In other words, the latency for a given media instance x between two processes p_i and p_j must be kept constant to be Δ legal. We call such a constant latency $\delta_{x,i,j}$.

Simultaneity

With Δ legality, we have specified temporal relations among events coming from the same media instance. With the simultaneity, we specify temporal relation among each media instances.

In its "kinematical part" (Einstein, 1905) defines simultaneity as follows:

> If [...] I say "That train arrives here at 7 o'clock". I mean something like this: "The pointing of the small hand of my watch to 7 and the arrival of the train are simultaneous events"

This definition of simultaneity fits well with highly synchronous interactions. For example, in a racing game, it is important to have simultaneous events: "when a car A crosses the finish line, the car B is 2 meters behind". We can understand it like this: "the car A is finishing and the car B is 2 meters behind the finish line are simultaneous events".

In the next part, "on the relativity of lengths and times", (Einstein, 1905) shows that a human can perceive two events simultaneously only in its own local time system, and that common perception of time is possible only among close persons.

An example is given figure 3: if $d_1 = d_1'$ and $d_2 = d_2'$ then events occur on the same order for p_1 and p_2 and relative times are respected at the playout. Suppose that $w_2(x)u$ and $w_1(x)v$ are positions of the cars (at their respective time) in a racing game. The fact that updates occur with the same relative times ($d_1 = d_1'$) allows the systems of p_1 and p_2 to detect identically a collision (according to the spacial positions u and v and to the delay d_1).

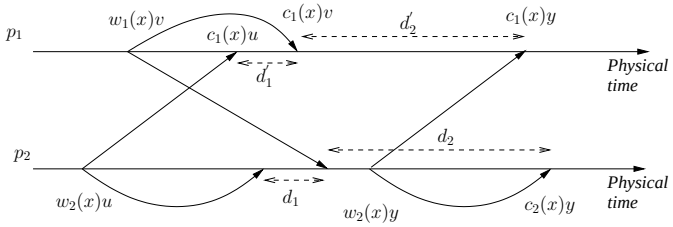

Fig. 3. *The simultaneity criterion*

Formally, in a history $\widehat{H}$ $c_j(x_k)v$ and $c_j(x_l)u$ are simultaneous if:
$\forall p : t(c_j(x_k)v) - t(c_j(x_l)u) = t(c_p(x_k)v) - t(c_p(x_l)u)$

Then, we define a history $\widehat{H}$ as simultaneous if:
$\forall p \forall v, u \forall k, l$: $c_p(x_k)v$ and $c_p(x_l)u$ are globally simultaneous.

Perceptive Consistency and Conflicts

We define the *perceptive consistency* with two criteria: the *simultaneity* and the Δ *legality*. In this section, we show that a perceptively consistent history $\widehat{H}$ is free of conflicts.

If two events are dated by the same physical clock, then they are ordered:
$t_j(c_j(x)v) < t_j(c_j(x)u) \Leftrightarrow c_j(x)v \rightarrow_j c_j(x)u$
Then, from the definition of the simultaneity, conflicts are avoided:
$\forall c_j(x)v, c_j(x)u \in H \ \forall k : (c_j(x)v \rightarrow_j c_j(x)u) \Leftrightarrow (c_k(x)v \rightarrow_k c_k(x)u)$
However, the Δ legality specifications allows to run two *handle* operations at the same local date, as the following shows:
$\delta_{x,i,j} + T(w_i(x)v) = \delta_{x,k,j} + T(w_k(y)u)$
$\Rightarrow T(c_j(x)v) = T(c_j(y)u)$
$\Rightarrow t_j(c_j(x)v) = t_j(c_j(y)u)$
Where $\delta_{x,i,j}$ and $\delta_{x,k,j}$ are constants used to specify the network latency, and T the universal clock (cf definition of the Δ legality). In this case, conflicts can be avoided by giving the priority to the operation which have the smaller writer's id (as in the (Mauve *et al.*, 2004) protocol). This allows replicas to order well operations without message passing.

```
1  int vector_myid[n], initialized with 0
2  while provisioning
3    receive an artificial update (u, t_i(u)) from p_i
4      if currentLocalTime − t_i(u) > vector_myid[i]
5      then vector_myid[i] := currentLocalTime − t_i(u)
6  endwhile
7  Broadcast vector_myid[n]
```

Fig. 4. Calculation of the local vector

```
8     sync = false  variable to stop the calculation
9     dif_temp[n]
10    Receive the n vectors
11    While sync != true
12       Chose p_ref as max(p_x) where 1 ≤ x ≤ n and
            p_x has never been chosen
13       For i from 1 to n
14          p_ajust := p_i
15          For j from 1 to n
16             dif_temp[p_j] :=
                 vector_j[p_ref] − vector_j[p_ajust]
17          EndFor
18          dif[p_i] := max_j(dif_temp[p_j])
19       EndFor;
20       If ∀i ∈ [1...n] dif[p_i] > 0 then
21         sync := true
22         for i from 1 to n
23           ajust[p_i] := dif[p_i]−
               (vector_myid[p_ref] − vector_myid[p_i])
24         EndFOr
25       EndIf
26    EndWhile
```

Fig. 5. Calculation of the local lags

```
28  while(MessagesAreSuposedToArrive)
27       receive (u, t_i(u)) from p_i
28       play u when
29          t_myid == t_i(u) + vector_myid[i] + ajust[p_i]
30       EndWhile
```

Fig. 6. The updates processing

THE ALGORITHM

The proposed algorithm provides a state compliant to the perceptive consistency (i.e. Δ legal and the simultaneous). It is composed of three phases: the calculation of the local vector, the calculation of the local lag and the messages processing.

The two first phases are part of the initialization. The **calculation of the local vector** (figure 4) is an estimation of the latencies between the local replica and the other. $vector_{myid}[n]$ is the vector that saves the value measured: it contains absolute differences between the perception of the remote clock and the local clock. It can be seen as an estimation if the real latencies (if the clocks are synchronized with NTP).

Then each replica broadcasts regularly some artificial events ($(u, t_i(u))$ where u is the artificial update coming from p_i and sent at $t_i(u)$, a value measured by the p_i's clock). $Provisioning$ is a variable used to control the duration of the calculation. During this duration, several updates are received from each p_i but only the higher difference is kept in order to take avoid the effect of the network jitter. Then $vector_{myid}[i]$ contains the latencies considered as the maximum latency from p_i to p_{myid}. Our algorithm is optimistic as network latency can grow through the time. However in good network conditions, we consider that next messages will arrive before these delays. Finally, when the vector is computed, it is send to the group (line 7).

With vectors and for each source, each replica can compute the lags to introduce locally (before playing out the updates) in order to obtain the simultaneity property: this is the goal of the **calculation of the local lags**.

When a user's process receives the n vector (line 8), it becomes aware of the temporal perception that each replica have of the other. The algorithm compare each "temporal perceptions" (the vectors) and adjusts the local lags according to the process which receives updates with the higher latency.

p_{ref} is the "reference process" (line 12) used to compute the clocks differences (line 16). For each media instances, the maximum difference is computed (line 18) and stored in table $dif[]$. However, a process can be chosen as reference only if its clock is always perceived late by another (line 20). If not, the lag to introduce would be negative (line 23). Thus the algorithm ends when each process is supposed to add a positive lag.

If the current p_{ref} fails the test line 20, another process is chosen. In order the decide on the same value, the computation tests processes as reference from the bigger id to the smaller to finish when the reference process satisfy the "if" line 20. The next section provide a proof of the termination.

When the computation arrives at line 22, table $dif[p_i]$ contains the differences of the worst receivers, i.e. for each media instances the ones which have the higher latency. Then in line 23, the local process compute the local delays to add to be as delayed as the worst receiver.

When the table $ajust[]$ is computed, the replica processes the received update as shown by figure 6.

TERMINATION OF THE ALGORITHM

The fact that one of the involved processes is a successful reference (p_{ref}) need to be demonstrated. More

precisely: it exits a process, when chosen as the reference process, that satisfies the formula $\forall i dif[pp_i] > 0$ with $1 < i < n$ and $i \neq ref$ (line 20). In other words, each processor p_i ($1 < i < n$ and $i \neq ref$) is measured at least once late compared to p_{ref}. We exclude the comparison between the processor reference with itself, because $dif[p_{ref}]0$ is always null.

Definition 1: Let the local precedence $f(i,j) = i \xrightarrow{k} j$ be the binary relation such as from the processor p_k point of view p_i is measured late compared to the processor p_j. Otherwise if p_k measures a null delays between p_i and p_j, then $i > j \Rightarrow i \xrightarrow{k} j$

f is anti-reflexive ($f(i,i)$ is undefined) and antisymmetric:

$$\neg(i \xrightarrow{k} j) \Leftrightarrow j \xrightarrow{k} i \tag{1}$$

Additionally:

$$\forall k(\neg(i \xrightarrow{k} j)) \Rightarrow \forall k(j \xrightarrow{k} i) \tag{2}$$

Moreover from the p_k point of view, if p_i is late compared to p_j, which is itself late compared to p_l, then p_i is late compared to p_l, i.e. f is transitive:

$$(i \xrightarrow{k} j) \wedge (j \xrightarrow{k} l) \Rightarrow i \xrightarrow{k} l$$

Additionally, from the conjunction property of the $\forall$ quantifier:

$$\forall k(i \xrightarrow{k} j) \wedge \forall k(j \xrightarrow{k} l) \Rightarrow \forall k(i \xrightarrow{k} l) \tag{3}$$

Theorem 1: In the local lags calculation it exists p_i ($1 \leq i \leq n$) that satisfies the formula $\forall i \in [1 \ldots n]\ dif[p_i] > 0$.
With the *local precedence* relation the theorem (1) is specified as follows (p_j has been chosen as a successful reference):

$$\exists j \forall i \neq j \exists k(i \xrightarrow{k} j) \tag{4}$$

Proof 1: Theorem (1) is proved if (4) is true. We proceed by reducing it to the absurd, i.e. we prove that the following formula (5) is false:

$\neg$ (4) $\Rightarrow \neg(\exists j \forall i \exists k(i \xrightarrow{k} j)) \Rightarrow \forall j \exists i \forall k \neg(i \xrightarrow{k} j)$
Thus, from the formula (1):

$$\Rightarrow \forall j \exists i \forall k(j \xrightarrow{k} i) \tag{5}$$

Let us suppose that (5) is true, then:
$j = x_1 \Rightarrow \exists i = x_2 : \forall k(x_1 \xrightarrow{k} x_2)$
Similarly, $j = x_2 \Rightarrow \exists i = x_3 : \forall k(x_2 \xrightarrow{k} x_3)$
And generally, $\forall k(x_{n_p} \xrightarrow{k} x_{n_p+1})$ ($j = x_{n_p}$ and $i = x_{n_p+1}$)
Thus:

$$\forall k(x_1 \xrightarrow{k} x_2) \wedge \ldots \wedge \forall k(x_{n_p} \xrightarrow{k} x_{n_p+1}) \tag{6}$$

In (6) for the local precedence binary relation we have n_p left members and n_p right members. But x_i goes from x_1 to $x_{n_p} + 1$, we found a member that appears as a right member and as a left member. Let $x_a = x_b$ be this member.

If $\forall k(x_n \xrightarrow{k} x_{n+1})$ $(n \in [1 \ldots n_p] \overset{def}{\Rightarrow} x_n \neq x_{n+1}$
Thus it exists x_c such as:

$$\forall k(x_a \xrightarrow{k} x_{a+1}) \wedge \ldots \wedge \forall k(x_{c-1} \xrightarrow{k} x_c) \wedge \forall k(x_c \xrightarrow{k} x_{c+1}) \wedge \ldots \wedge \forall k(x_{b-1} \xrightarrow{k} x_b)$$

In this case, we apply the property (3) to (6) with x_a, x_b and x_c:

$$\forall k(x_a \xrightarrow{k} x_c) \wedge \forall k(x_c \xrightarrow{k} x_b)$$

As $x_a = x_b$, if is a contradiction to the property (2):

$$\forall k(x_a \xrightarrow{k} x_c) \wedge \forall k(x_c \xrightarrow{k} x_a) \tag{7}$$

Then (5) is false
We have proved the theorem (1), i.e. the local lags calculation ends with a success.

CONCLUSION AND OUTLOOK

In this paper we investigate the problem of consistency in highly interactive and fully distributed video games when coupled with a dead reckoning mechanism. We provide a consistency model based on user requirement, i.e. local operations are instantaneous and conflicts are discarded thank to the simultaneity property. Then we provide an optimistic consistency protocol that permits to maintain a consistent state as fast as the network allows, and for both discrete and continuous media.

Our consistency model and our protocol show formally that time sensitive interactions are subject to the network latencies. Indeed inconsistencies are detected after an amount of time closely related to the real network delay and repaired with a rollback-like mechanism. It formally shows that dead-reckoning techniques with our protocol are efficient when the network delays are close to the user requirement. Thus it formally shows that a network providing latencies lesser than l is able to hide latencies for interactivities constrained by a response time bigger than l. Otherwise, when games run over a high latency network (as mobile games), additional or other tricks must be used to compensate the network delays.

The protocol was already tested in the continuous domain for Networked Musical Performance (Bouillot,

2003) on MAN where conflicts cannot appear but not yet in the discrete domain. Then we plan to test it in an interactive game to enhance the conflict resolution delays and particularly for contacts resolutions.

REFERENCES

Aggarwal, Sudhir, Banavar, Hemant, Khandelwal, Amit, Mukherjee, Sarit, & Rangarajan, Sampath. 2004. Accuracy in dead-reckoning based distributed multi-player games. *Pages 161–165 of: SIGCOMM 2004 Workshops: Proceedings of ACM SIGCOMM 2004 workshops on NetGames '04*. New York, NY, USA: ACM Press.

Akyildiz, I., & Yen, W. 1996. Multimedia Group Synchronization Protocols for Integrated Services Networks. *IEEE Jounal on selected area in communications*, Vol. 14(No. 1), p. 162.

Bouillot, N. 2003 (October). Un algorithme d'auto synchronisation distribuee de flux audio dans le concert virtuel reparti. *In: Proc. of The Conference Francaise sur les Systemes d'Exploitation (CFSE'3)*.

Bouillot, Nicolas, & Gressier-Soudan, Eric. 2004. Consistency models for distributed interactive multimedia applications. *SIGOPS Oper. Syst. Rev.*, Vol. 38(4), 20–32.

Cronin, E., Filstrup, B., Kurc, A., & Jamin, S. 2002. An efficient Synchronization Mechanism for Mirrored Game Architectures. *In: ACM Netgames'02*.

Einstein, A. 1905. On the electrodynamics of moving bodies. *Annalen der Physik*, Vol. 17(June). translated from german.

Gautier, L., & Diot, C. 1998. Design and Evaluation of MiMaze, a Multi-Player Game on the Internet. *Pages 233–236 of: International Conference on Multimedia Computing and Systems*.

Mauve, M., Vogel, J., Hilt, V., & Effelsberg, W. 2004. Local-lag and Timewarp: Providing Consistency for Replicated Continuous Applications. *IEEE Transactions on Multimedia*, Vol. 6(Nr. 1).

Mills, D. 1989 (Sept.). *Network Time Protocol (version 2) specification and implementation*. Network Working Group Request for Comments: 1119.

Pantel, L., & Wolf, L. C. 2002. On the suitability of dead reckoning schemes for games. *Pages 79–84 of: Proceedings of the 1st workshop on Network and system support for games*. ACM Press.

Raynal, M., & Schiper, A. 1996 (Sept.). A Suite of Formal Definitions for Consistency Criteria in Distributed Shared Memories. *Pages 125–130 of: Proceedings Int Conf on Parallel and Distributed Computing (PDCS'96)*.

Steinmetz, R. 1996. Human Perception of Jitter and Media Synchronization. *IEEE Jounal on selected area in communications*, Vol. 14(No. 1), p. 61.

Zhou, S., Cai, W., Turner, S. J., & Lee, F. B. S. 2002. Critical causality in distributed virtual environments. *Pages 53–59 of: Proceedings of the sixteenth workshop on Parallel and distributed simulation*. IEEE Computer Society.